Friedrich-Alexander-Universität
Technische Fakultät

Dissertation

2026

Dominic Lohr

Feedback-Strategien für Programmierlernsysteme:
Von empirischen Analysen zur automatisierten Bereitstellung

Feedback-Strategien für Programmierlernsysteme: Von empirischen Analysen zur automatisierten Bereitstellung

Der Technischen Fakultät
der Friedrich-Alexander-Universität
Erlangen-Nürnberg
zur Erlangung des Doktorgrades
Dr.-Ing.
vorgelegt von

Dominic Lohr
aus
Nürnberg

Als Dissertation genehmigt von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der mündlichen Prüfung: 05. März 2026

Gutachter: Prof. Dr. Marc-Pascal Berges
Prof. Dr. Sven Strickroth

Kurzfassung

Algorithmische Denk- und Handlungsweisen gelten zunehmend als Schlüsselkompetenzen in einer von Software- und Datenstrukturen geprägten Gesellschaft. Programmieren wird dabei nicht nur als technische Fertigkeit, sondern als zentrales Mittel zur Förderung dieser Kompetenzen verstanden. Das Erlernen von Programmieren unterstützt zentrale Aspekte des Computational Thinking, insbesondere analytisches Problemlösen, Mustererkennung und die Zerlegung komplexer Aufgaben in algorithmisch lösbare Teilprobleme, ist jedoch mit erheblichen Herausforderungen verbunden. Empirische Übersichtsarbeiten verweisen auf persistente Hürden beim Einstieg in die Programmierung, die zu hohen Abbruchraten in entsprechenden Studiengängen beitragen. Um Lernprozesse wirksam zu fördern, sind daher differenzierte Begleitung und gezielte Rückmeldungen von besonderer Bedeutung. In der Bildungsforschung gilt *Feedback* als zentraler Einflussfaktor auf Lern- und Leistungsentwicklungen. Im Kontext des Programmierenlernens kann es nachweislich dazu beitragen, Fehlkonzepte zu korrigieren, Denkstrategien zu fördern und Lernprozesse zu stabilisieren. Offene Forschungsfragen bestehen jedoch hinsichtlich der Bedingungen und Kontexte, unter denen Feedback im Programmierlernprozess tatsächlich lernwirksam ist. Unterschiedliche Definitionen, Zielsetzungen und Untersuchungskontexte erschweren bislang die Vergleichbarkeit von Studien und deren Übertragbarkeit auf konkrete Lehrsituationen. Vor diesem Hintergrund gewinnen sogenannte *Feedback-Strategien* – abgestimmte Kombinationen von Inhalt, Zeitpunkt, Funktion und Form von Feedback – an Bedeutung, da sie eine systematische und kontextsensible Gestaltung sowie Erforschung von Feedbackprozessen ermöglichen. Eine zentrale Herausforderung besteht darin, solche Strategien auch in digitalen Lernumgebungen umzusetzen. Programmierlernsysteme versuchen, Lernende durch automatisiertes Feedback zu unterstützen, jedoch zeigen systematische Untersuchungen, dass sich dieses in der Praxis meist auf einfache Korrektheitsinformationen oder generische Hinweise beschränkt und individuelle Lernkontexte nur unzureichend berücksichtigt werden. Damit bleibt das Potenzial adaptiver Lernumgebungen zur Bereitstellung kontextsensitiven Feedbacks bislang weitgehend ungenutzt.

Hier setzt die vorliegende Dissertation an und verfolgt das Ziel, Feedback-Strategien beim Programmierenlernen didaktisch zu fundieren und technologisch zu operationalisieren. Ausgangspunkt ist eine empirische Analyse der Entscheidungslogiken erfahrener Lehrpersonen, die zeigt, dass professionelles Feedback nicht primär der Fehlerkorrektur dient, sondern als situativ begründete Intervention verstanden werden sollte, welche Produkt- und Prozessindikatoren integriert. Um solche didaktischen Intentionen mit maschinenlesbaren Zustandsbeschreibungen zu verknüpfen und damit die Grundlage regelgeleiteter Feedbackentscheidungen zu bilden, werden anschließend zwei Modellbausteine hergeleitet: das *Y-Modell* zur systematischen Beschreibung von Aufgabenkontexten sowie das Konzept der *Antwortklassen* als diagnostisches Beschreibungssystem zur kategorialen Erfassung und Interpretation von Lernendenantworten. Um kontextsensitives Feedback nicht nur konsistent, sondern auch *skalierbar* bereitzustellen, wird anschließend untersucht, inwieweit sich große Sprachmodelle (LLMs) eignen, auf Basis dieser Strukturen qualitativ hochwertiges, situationsangemessenes Feedback zu generieren. Hierfür werden zwei Studien durchgeführt, deren Befunde zeigen, dass präzise definierte Aufgaben- und Antwortkontexte die Relevanz und Kohärenz der von LLMs generierten Rückmeldungen deutlich verbessern. Sie verdeutlichen zugleich, in welchem Umfang sich verschiedene Feedback-Typen automatisiert erzeugen lassen und welche Grenzen in Bezug auf Konsistenz, Nachvollziehbarkeit und didaktische Steuerbarkeit bestehen. Aufbauend darauf wird mit `APFEL` ein konzeptueller Systementwurf vorgestellt, der die gewonnenen Erkenntnisse integriert, regelbasierte mit datengetriebenen Ansätzen verbindet und exemplarisch aufzeigt, wie kontextsensitive Feedbackentscheidungen in Programmierlernsystemen transparent, adaptiv und didaktisch steuerbar umgesetzt werden können.

Die Arbeit leistet damit einen Beitrag zur theoretischen Fundierung, empirischen Beschreibung und technologischen Umsetzung kontextsensitiver Feedbackprozesse. Sie zeigt, wie didaktische Modelle, empirische Evidenz und KI-gestützte Verfahren zusammengeführt werden können, um Lernprozesse beim Programmierenlernen gezielt zu unterstützen und deren systematische Erforschung in digitalen Lernumgebungen zu fördern.

Abstract

Algorithmic thinking and practices are increasingly regarded as key competencies in a society shaped by software and data structures. Programming is therefore not only understood as a technical skill but also as a central means of fostering these competencies. Learning to program supports key aspects of computational thinking – in particular analytical problem solving, pattern recognition, and the decomposition of complex tasks into algorithmically solvable subproblems – yet it remains associated with considerable challenges. Empirical reviews highlight recurring difficulties when entering programming, which contribute to high dropout rates in related degree programs. To effectively promote learning processes, differentiated guidance and targeted feedback arey of particular importance. In educational research, *feedback* is considered a key factor influencing learning and performance development. In the context of learning to program, it can demonstrably help to correct misconceptions, promote thinking strategies, and stabilize learning processes. However, open research questions remain concerning the conditions and contexts under which feedback is truly effective in programming education. Divergent definitions, goals, and research contexts have so far hindered the comparability of studies and the transferability of findings to concrete teaching situations. Against this background, so-called *feedback strategies* – coordinated combination of content, timing, function, and form of feedback – are gaining importance, as they enable a systematic and context-sensitive design and investigation of feedback processes. A key challenge lies in implementing such strategies within digital learning environments. Programming learning systems attempt to support learners through automated feedback, but systematic reviews reveal that this feedback is often limited to simple correctness information or generic hints and insufficiently accounts for individual learning contexts. Consequently, the potential of adaptive learning environments to provide context-sensitive feedback remains largely untapped.

This dissertation addresses this gap by aiming to both didactically ground and technologically operationalize feedback strategies in programming education. It begins with an empirical analysis of the decision-making logics of experienced teachers, showing that professional feedback is not primarily about error correction but rather constitutes a situationally justified intervention that integrates both product and process indicators. To link such educational intentions with machine-readable state descriptions – and thus form the basis for rule-guided feedback decisions – two model components are developed: the *Y-Model* for the systematic description of task contexts, and the concept of *answer classes* as a diagnostic framework for the categorical representation and interpretation of learners' answers. To provide context-sensitive feedback that is not only consistent but also *scalable*, the study further investigates the extent to which Large Language Models (LLMs) can generate high-quality, contextually appropriate feedback based on these structures. Two studies demonstrate that precisely defined task and answer contexts significantly enhance the relevance and coherence of LLM-generated feedback. They also reveal to what extent various feedback types can be generated automatically, as well as the limitations in terms of consistency, transparency, and didactic controllability. Building on these insights, the dissertation introduces `APFEL` – a conceptual system design that integrates empirical findings, combines rule-based and data-driven approaches, and exemplifies how context-sensitive feedback decisions can be implemented in programming learning systems in a transparent, adaptive, and pedagogically controllable manner.

In doing so, the work contributes to the theoretical foundation, empirical characterization, and technological implementation of context-sensitive feedback processes. It demonstrates how educational models, empirical evidence, and AI-based approaches can be integrated to purposefully support learning processes in programming education and to advance their systematic investigation within digital learning environments.

Nutzung von Hilfsmitteln

Bei der Erstellung dieser Dissertation hat der Autor die in Tabelle 1 aufgeführten Applikationen als Unterstützungssysteme verwendet. Die spezifische Verwendung und der Zweck der Anwendung der Applikationen während des Schreibprozesses der Dissertation sind in Tabelle 1 aufgeführt. Die beschriebenen Aufgaben wurden nicht ausschließlich mit diesen Werkzeugen gelöst, sondern diese wurden ergänzend und situationsabhängig zur zusätzlichen Unterstützung für die aufgeführten Zwecke eingesetzt und ihre Ergebnisse wurden kritisch überprüft sowie überarbeitet. Für die eigentliche wissenschaftliche Arbeit – die Auswertung und Diskussion der Daten sowie die Erstellung von Inhalten, (Design-)Ideen, Konzepten oder Texten – wurden keine generativen Systeme verwendet. Der Inhalt dieser Dissertation wurde vollständig vom Autor erstellt, der sichergestellt hat, dass alle vorgestellten Materialien seine eigenen Arbeiten und Forschungen widerspiegeln. Darüber hinaus sind alle Passagen der Dissertation, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, deutlich gekennzeichnet und mit einer Quellenangabe versehen.

Werkzeug	Verwendung
`GPT-3.5` [1]; `GPT-4` [1] `HAWKI` [2]	Bei der Erstellung dieser Dissertation wurden die Sprachmodelle `GPT-3.5` und `GPT-4` unterstützend eingesetzt, insbesondere zur Verbesserung der Lesbarkeit und sprachlichen Kohärenz sowie zur Konvertierung von Tabellen in LaTeX-Format. Die Nutzung diente ausschließlich der Effizienzsteigerung bei zeitaufwändigen Formatierungs- und Textoptimierungsaufgaben, die der Autor auch manuell hätte durchgeführt können. Alle zur Anwendung verwendeten Prompts wurden eigenständig vom Autor erstellt. Sämtliche generierten Inhalte wurden vom Autor sorgfältig überprüft, hinsichtlich inhaltlicher Korrektheit und Angemessenheit bewertet, gegebenenfalls überarbeitet und korrigiert.
`DeepL Translator` [3]	Für die Erstellung erster Übersetzungsfassungen von englischsprachigen Publikationen ins Deutsche wurde das Tool DeepL Translator verwendet. Ziel war es, den Übersetzungsprozess zu beschleunigen und effizienter zu gestalten, um anschließend sprachliche und fachliche Präzision manuell zu optimieren. Alle maschinell übersetzten Texte wurden vom Autor überarbeitet, hinsichtlich fachlicher Genauigkeit und sprachlicher Angemessenheit geprüft, angepasst und korrigiert.

[1] `https://platform.openai.com/docs/models`
[2] `https://hawki.ai.fau.de/`
[3] `https://www.deepl.com/translator`

Tabelle 1: Verwendete Unterstützungssysteme während der Erstellung der Dissertation

Vorabveröffentlichungen

Teile der Dissertation wurden bereits vor Einreichung wörtlich oder sinngemäß veröffentlicht. Die entsprechenden Referenzen sind in Tabelle 2 dem jeweiligen Kapitel der Arbeit zugeordnet.

Kapitel	Veröffentlichungen
Kapitel 4	Dominic Lohr et al. (Juli 2024a). „“Let Them Try to Figure It Out First” - Reasons Why Experts (Do Not) Provide Feedback to Novice Programmers”. In: *Proceedings of the 2024 Innovation and Technology in Computer Science Education (ITiCSE 2024)*. Bd. 1. Milan, Italy: ACM, S. 7. Dominic Lohr et al. (Okt. 2025a). „"Ignore These Errors for Now- How Experts Provide Feedback on Steps Novices Take Towards Solving Programming Problems”. In: *Proceedings of the ACM Global on Computing Education Conference 2025 Vol 1*. Gaborone Botswana: ACM, S. 149–155. In beiden Fällen wurden ca. 60 % des Textes vom Erstautor verfasst. Die Daten wurden zu gleichen Teilen mit den Koautoren erhoben und kodiert. Der Erstautor leitete den Kodierprozess sowie die Sitzungen zur Datenauswertung.
Kapitel 5	Dominic Lohr et al. (2025c). „The Potential of Answer Classes in Large-scale Written Computer-Science Exams – Vol.2”. In: *Commentarii informaticae didacticae*. Aachen: Jörg Desel, Simone Opel. 90 % des Textes wurde vom Erstautor verfasst. Das Modell wurde vom Erstautor entwickelt und in Absprache mit dem Zweitautor verfeinert. Dritt- und Viertautor unterstützten bei Fallstudie und Textüberarbeitungen. Dominic Lohr et al. (Aug. 2023). „The Y-Model - Formalization of Computer Science Tasks in the Context of Adaptive Learning Systems”. In: *2023 IEEE 2nd German Education Conference (GECon)*. Berlin, Germany: IEEE, S. 1–6. 80 % des Textes wurde vom Erstautor verfasst. Das Modell wurde vom Erstautor entwickelt und in Absprache mit dem Zweitautor verfeinert. Dominic Lohr und Marc Berges (2025). „Der Weg ist das Ziel – Ein skalierbarer Wechsel von summativem zu formativem Assessment in der Programmierausbildung mit Antwortklassen”. In: *Proceedings Seventh Workshop "Automatische Bewertung von Programmieraufgaben"(ABP 2025)*. Gesellschaft für Informatik e.V. 90 % des Textes wurden vom Erstautor verfasst. Der Zweitautor stand beratend zur Seite und überarbeitete Teile des Textes.
Kapitel 6	Natalie Kiesler, Dominic Lohr und Hieke Keuning (Okt. 2023). „Exploring the Potential of Large Language Models to Generate Formative Programming Feedback”. In: *2023 IEEE Frontiers in Education Conference (FIE)*. College Station, TX, USA: IEEE, S. 1–5. Ca. 60 % des Textes wurden vom Zweitautor verfasst. Die Daten wurden vom Zweitautor erhoben und von allen Autoren zu gleichen Teilen kodiert. Der Zweitautor leitete dabei den Kodierprozess sowie die Sitzungen zur Auswertung der Daten. Dominic Lohr, Hieke Keuning und Natalie Kiesler (Feb. 2025). „You’re (Not) My Type – Can LLMs Generate Feedback of Specific Types for Introductory Programming Tasks?” In: *Journal of Computer Assisted Learning* 41.1, e13107. Ca. 60 % des Textes wurden vom Erstautor verfasst. Die Daten wurden vom Erstautor erhoben und von allen Autoren zu gleichen Teilen kodiert. Der Erstautor leitete dabei den Kodierprozess sowie die Sitzungen zur Auswertung der Daten.
Kapitel 7	Dominic Lohr et al. (2024b). „Adaptive Learning Systems in Programming Education: A Prototype for Enhanced Formative Feedback”. In: *Proceedings of DELFI 2024*. Fulda: Gesellschaft für Informatik e.V., S. 549–554. 95 % des Textes wurden vom Erstautor verfasst. Die Idee und Implementierung des Prototypen stammen vom Erstautor. Der Zweitautor stand beratend zur Seite und hat Teile des Textes überarbeitet.

Tabelle 2: Vorabveröffentlichungen zu einzelnen Kapiteln

Inhaltsverzeichnis

Kapitel 1

Einleitung

Generative KI-Systeme sind in der Mitte von Gesellschaft, Wirtschaft und Bildung angekommen. Dementsprechend veröffentlichen Regierungen und Bildungsorganisationen Leitlinien, wie diese Technologien verantwortungsvoll in Lehre und Forschung einzusetzen sind (UNESCO 2025). Auch groß angelegte Unternehmensbefragungen dokumentieren einen rapiden Diffusionsprozess über zahlreiche Branchen hinweg (Singla et al. 2025).

Vor diesem Hintergrund haben generative Systeme nicht nur Innovations- und Produktivitätsdiskussionen geprägt, sondern auch grundlegende Fragen danach aufgeworfen, welche Kompetenzen Menschen künftig benötigen, um in einer zunehmend automatisierten Welt handlungsfähig zu bleiben. In diesem Zusammenhang erregte eine zugespitzte Aussage besondere Aufmerksamkeit: Nvidia CEO Jensen Huang formulierte im Februar 2024 die These, man solle Kindern nicht mehr „Programmieren" beibringen, weil Rechenmaschinen zunehmend in der Lage seien, natürliche Sprache in lauffähige Programme zu übersetzen.[1] Diese Aussage macht exemplarisch deutlich, dass die zunehmende Leistungsfähigkeit generativer Systeme die Frage aufwirft, welche Rolle das *Programmierenlernen* künftig einnehmen sollte und welche Kompetenzen in einer KI-geprägten Welt langfristig relevant bleiben.

Im weiteren Diskurs ist jedoch eine klare begriffliche Unterscheidung zwischen *Coding* und *Programmieren* hilfreich, da beide Begriffe – sowohl in Forschung als auch in schulischer Praxis – häufig synonym oder unscharf verwendet werden (Corradini, Lodi und Nardelli 2018). Unter *Coding* kann eng gefasst das Übersetzen vorliegender Lösungsentwürfe in syntaxkonforme Quelltexte verstanden werden. *Programmieren* umfasst demgegenüber den gesamten *Problemlöseprozess*: vom Analysieren und Modellieren über das Entwerfen von Daten- und Kontrollstrukturen, das Testen und Debuggen bis zur Reflexion von Qualitätsmerkmalen wie Korrektheit, Robustheit, Effizienz und Wartbarkeit (Robins, Rountree und Rountree 2003; Denning und Tedre 2019). In diesem breiten Verständnis ist Programmieren nicht bloß eine technische Fertigkeit, sondern eine Ausdrucks- und Gestaltungsform formaler Problembeschreibung.

Aus fachdidaktischer Perspektive kann somit argumentiert werden, dass Programmieren auch unter Bedingungen leistungsfähiger generativer Systeme *gebraucht* wird und zwar mehr denn je: Erstens erfordert der kompetente Umgang mit generativen Systemen ein ausgeprägtes *Modellierungs- und Beurteilungsvermögen*, da Nutzerinnen und Nutzer KI-generierte Inhalte analysieren, bewerten und iterativ verfeinern müssen. Dieser Prozess des Formulierens, Prüfens und Überarbeitens von Eingaben entspricht zentralen Komponenten des *computational thinking* wie Abstraktion, Evaluation und Iteration (Hsu 2025). Zweitens kann Programmieren im Sinne eines

[1] Vgl. die offizielle Mitschrift/Wiedergabe des Auftritts von Jensen Huang beim World Government Summit 2024 sowie `https://www.computing.co.uk/news/4179836/dont-encourage-kids-code-nvidia-ceo`

digitalen Schreibens als genuine Kulturtechnik verstanden werden, die gesellschaftliche Teilhabe ermöglicht, weil sie produktive Handlungsfähigkeit in einer softwarevermittelten Welt schafft (Vee 2017; Bajohr und Krajewski 2024; Coy 2008). Diese Bedeutung hat auch angesichts generativer Systeme nicht abgenommen: Wie Bajohr und Krajewski (2024) hervorheben, bleibt das Programmieren eine Voraussetzung, um algorithmische Strukturen kritisch zu verstehen, zu kommentieren und weiterzuschreiben und damit digitale Kultur nicht nur zu nutzen, sondern aktiv mitzugestalten. Drittens verweist die weitreichende Durchdringung von Wirtschaft und öffentlichem Leben mit Software- und Dateninfrastrukturen darauf, dass algorithmische Denk- und Handlungsweisen *Schlüsselkompetenzen* darstellen (Futschek 2006; Byrka et al. 2021).

Der Erwerb von Programmierkompetenzen ist jedoch nachweislich mit erheblichen Herausforderungen verbunden (Becker et al. 2023). Synthesen der empirischen Forschung berichten über persistente Hürden bei Syntax, Semantik, Abstraktion, Kontrollfluss, Rekursion und Fehlersuche (Lahtinen, Ala-Mutka und Järvinen 2005; Qian und Lehman 2018). Metastudien und Querschnittsanalysen zeigen zudem erhöhte Durchfall- und Abbruchquoten in Einführungsveranstaltungen, deren Höhe in Abhängigkeit von Kursgröße, Institutionstyp und Bildungssystem deutlich variiert (Watson und Li 2014; Bennedsen und Caspersen 2019). Hinzu kommt eine wachsende *Leistungsheterogenität* unter Programmieranfängerinnen und -anfängern. Ein wesentlicher Treiber ist stark divergierende *Vorerfahrungen*, die über die ersten Studienphasen hinweg zu stabilen Leistungsdifferenzen führen können (Bui et al. 2023). Während erfahrene Programmiererinnen und Programmierer über abstrahierte Wissensschemata, planungsorientierte Strategien und effektive Verfahren der Fehlersuche verfügen, arbeiten Einsteigerinnen und Einsteiger häufig mit oberflächlich organisiertem Wissen und unsystematischen, zeilenbasierten Vorgehensweisen. Sie sind daher in besonderem Maße auf gezielte Unterstützung beim Planen, Testen und Debuggen angewiesen (Robins, Rountree und Rountree 2003).

Diese unterschiedlichen Ausgangsbedingungen erschweren es, Lernende mit einheitlichen Lehr- und Unterstützungsformen gleichermaßen zu erreichen. Entsprechend steigt der Bedarf an *adaptiven* Ansätzen, die an das individuelle Vorwissen und den jeweiligen Lernfortschritt anschließen. Empirische Studien zeigen, dass insbesondere adaptive formative Assessments dazu beitragen können, den Lernprozess von Programmieranfängerinnen und -anfängern gezielter zu fördern, indem sie Rückmeldungen an den Bearbeitungsstand anpassen, häufige Verständnisprobleme aufgreifen und so das Erlernen zentraler Konzepte unterstützen (Thangaraj, Ward und O'Riordan 2024).

Feedback gehört zu den wirksamsten Einflussfaktoren auf Lernprozesse – sofern es inhaltlich gehaltvoll, aufgabenbezogen und prozessunterstützend gestaltet ist (Hattie und Timperley 2007; Wisniewski, Zierer und Hattie 2020). Zugleich zeigen Befunde der allgemeinen Feedbackforschung, dass Effekte auf den Lernerfolg stark variieren und unter ungünstigen Bedingungen sogar negativ ausfallen können (Kluger und DeNisi 1996; Wisniewski, Zierer und Hattie 2020). *Was*, *wann* und *wie* rückgemeldet wird, ist damit keine triviale Gestaltungsfrage, sondern eine entscheidungs- und kontextabhängige didaktische Herausforderung. Bloom (1984) zeigte bereits in den 1980er Jahren, dass individuelles Tutoring im Durchschnitt eine Leistungssteigerung um zwei Standardabweichungen gegenüber herkömmlichem Unterricht bewirken kann – ein Effekt, der maßgeblich auf kontinuierliche Rückmeldung und gezielte Unterstützung zurückgeführt wird (Bloom 1984). Spätere Metaanalysen zeigen, dass die Wirksamkeit individueller Förderung insbesondere auf adaptivem Unterstützungsprozessen wie Feedback beruht, die Lernprozesse eng begleiten und Fehlkonzepte unmittelbar adressieren (Vanlehn 2011; Kulik und Fletcher 2016). Gerade in großen, heterogenen Lerngruppen stoßen *1:1-Formate*, die in empirischen Studien besonders hohe Lerneffekte zeigen, jedoch rasch an ihre organisatorischen Grenzen (Bloom 1984; Vanlehn 2011).

Da sich solche Formen personalisierter Interaktion in hochfrequenten Lehrsituationen kaum umsetzen lassen, richtet sich die Aufmerksamkeit zunehmend auf digitale Systeme, die vergleichbare Mechanismen technisch abbilden sollen. Im Bereich der *Intelligenten Tutoring-Systeme* und *Programmierlernumgebungen* wird untersucht, wie Lernende beim Entwickeln, Testen und Reflektieren von Code durch automatisierte Rückmeldungen begleitet werden können (Merrill et al. 1992; Keuning, Jeuring und Heeren 2019). Ziel dieser Systeme ist es, individualisiertes, kontextsensitives Feedback in großskaligen Lehrkontexten bereitzustellen und damit zentrale Prinzipien wirksamer Lerntutorinnen und -tutoren in skalierbarer Form nutzbar zu machen.

Automatisierte Umgebungen für Programmieraufgaben (z. B. Online-Judges, Autograder, ITS) sind daher seit Jahren verbreitet (Ala-Mutka 2005; Ihantola et al. 2010). Obwohl die Relevanz von Feedback weithin anerkannt ist, zeigen systematische Übersichtsarbeiten erhebliche Defizite in der Qualität und Wirksamkeit automatisierter Rückmeldungen (Keuning, Jeuring und Heeren 2019). Das Feedback bleibt häufig auf „richtig/falsch", Testausgaben oder generische Hinweise beschränkt. Die automatisierten Rückmeldungen beschränken sich überwiegend darauf, *Fehler zu identifizieren*, aber seltener *nächste Schritte* anzuleiten oder *kontextspezifische Begründungen* zu liefern. Zudem ist die didaktische Anpassbarkeit für Lehrende oft begrenzt (Keuning, Jeuring und Heeren 2019). Zugleich mangelt es an belastbaren Wirksamkeitsnachweisen für Lerngewinne in authentischen Settings.

Erfahrene Lehrpersonen hingegen sind in der Lage – wenn auch mit einem gewissen Zeitaufwand – qualitativ hochwertiges, elaboriertes Feedback zu geben, das auf den individuellen Lernstand und die spezifische Aufgabenstellung zugeschnitten ist (Brandmo und Gamlem 2025). In der Feedbackforschung werden solche abgestimmten Formen der Rückmeldung als *Feedback-Strategien* bezeichnet – koordinierte Pläne, die Inhalt, Zeitpunkt, Funktion und Präsentationsform von Feedback aufeinander abstimmen, um Lernprozesse gezielt zu unterstützen (Narciss 2007). Allerdings existiert bislang kein Konsens darüber, *wann*, *wie* und *welches* Feedback beim Programmierenlernen am wirksamsten ist (Ditton 2014).

Aus dieser Ausgangslage ergibt sich der Anspruch, menschliche Feedback-Strategien *empirisch* zu erfassen, *formal* zu beschreiben und *technologisch* so umzusetzen, dass sie in realen Lernumgebungen *skalierbar, kontextsensitiv und nachvollziehbar* bereitgestellt werden können. Die Arbeit verfolgt damit das Ziel, eine Brücke zwischen didaktischer Theoriebildung, empirischer Beobachtung und technologischer Umsetzung zu schlagen. Sie leistet zugleich einen Beitrag zur Grundlagenforschung im Bereich der kontextsensitiven Lernunterstützung und zur Gestaltung adaptiver Lernsysteme für das Programmierenlernen. Im Zentrum steht die leitende Forschungsfrage, *wie Feedback-Strategien so beschrieben und formalisiert werden können, dass sie als Grundlage für die automatisierte Bereitstellung kontextsensitiven Feedbacks in Programmierlernsystemen dienen können.* Die Beantwortung dieser Frage erfolgt schrittweise entlang vier aufeinander aufbauender Untersuchungsphasen, die jeweils in einem eigenen Kapitel dargestellt sind:

Kapitel 4 untersucht zunächst, **auf welcher Grundlage erfahrene Lehrpersonen entscheiden, ob und wann sie Feedback geben und wie sie dieses ausgestalten**. Auf Basis einer Expertenstudie wird ein empirisch fundiertes Kategoriensystem entwickelt, das typische Entscheidungslogiken und Feedbacktypen beschreibt. Dieses Kategoriensystem bildet die Grundlage für die Modellierung adaptiver Feedbackprozesse.

Kapitel 5 überführt diese empirischen Befunde in ein **formales Bezugsmodell zur Beschreibung von Aufgaben, Antworten und Kontextfaktoren**. Dabei wird das *Y-Modell* eingeführt, das die strukturellen Beziehungen zwischen Aufgabenanforderungen, Lernendenantworten und Feedbackmöglichkeiten beschreibt. Zentrale Rolle spielt hierbei das Konzept der *Antwortklassen*, mit dem typische Reaktionsmuster von Lernenden systematisch kategorisiert werden können.

Kapitel 6 prüft anschließend die **technologische Umsetzbarkeit kontextsensitiven Feedbacks durch große Sprachmodelle (LLMs)**. In zwei Studien wird untersucht, inwieweit LLMs in der Lage sind, unterschiedliche Feedback-Typen zu erzeugen, und wie sich die Qualität der Rückmeldungen unter Variation des bereitgestellten Kontexts verändert. Die Ergebnisse erlauben eine differenzierte Einschätzung der Chancen und Grenzen generativer Systeme im Bildungskontext.

Kapitel 7 bündelt schließlich die theoretischen und empirischen Ergebnisse in einem konzeptuellen **Systementwurf** namens `APFEL` (Adaptives Programmier-Feedback für E-Learning). Dieses zeigt exemplarisch, wie empirisch begründete und formal modellierte Feedback-Strategien in Programmierlernsystemen integriert werden können. Das Design kombiniert regelbasierte und datengetriebene Verfahren und demonstriert damit einen Ansatz, der didaktische Nachvollziehbarkeit mit technologischer Skalierbarkeit verbindet.

Kapitel 2

Feedback in Lernsystemen

2.1 Feedback

Feedback nimmt eine zentrale Rolle in Bildungsprozessen ein und zählt zu den am häufigsten untersuchten Interventionen in Lehr-Lern- und Arbeitskontexten (Ditton 2014). Aufgrund seiner vielfach empirisch belegten Wirksamkeit gilt es als eines der wirkungsvollsten Instrumente zur Förderung von Lern- und Leistungsentwicklungen (Narciss 2017; Hattie und Timperley 2007; Kluger und DeNisi 1996).

Insbesondere die Metaanalyse von Hattie (2009) verdeutlicht, dass qualitativ hochwertiges Feedback einen maßgeblichen Einfluss auf den Lernfortschritt ausüben kann – teils wird es sogar als *wichtigste* Intervention postuliert, um Lernen zu fördern (Hattie und Timperley 2007). Trotz der breiten Forschungslage bestehen jedoch oft Unklarheiten hinsichtlich grundlegender Begriffsverwendungen, systematischer Klassifikationen sowie didaktischer Implikationen verschiedener Feedbackformen, da diese je nach Kontext stark variieren können.

2.1.1 Begriffsbestimmung und Abgrenzung

Der Begriff *Feedback* stammt aus dem Englischen und wurde in den deutschen Sprachgebrauch übernommen. Zwar lässt er sich allgemein mit *Rückmeldung* übersetzen, doch variiert die Bedeutung je nach disziplinärem Kontext mitunter erheblich:

In der Forschungsrichtung `Kybernetik` bezeichnet *Feedback* eine Form der *Rückkopplung*, bei der die Ausgangsgröße eines Systems rückwirkend auf dessen Eingangsgröße einwirkt. Im Zentrum steht hier die zielgerichtete Steuerung technischer, biologischer oder sozialer Systeme (Wiener 2019). In der `Robotik` hingegen wird *Feedback* häufig im Sinne sensorischer oder haptischer *Rückmeldungen* verwendet, die zur Unterstützung der Interaktion von Menschen mit Maschinen eingesetzt werden – beispielsweise durch visuelle oder akustische Signale bei der Bedienung technischer Systeme (Westervelt et al. 2007). In wirtschaftsnahen Fachdisziplinen wie z. B. dem `strategischen Management` wiederum steht *Feedback* meist für die *Bewertung* oder *Beurteilung* von Leistungen oder Verhaltensweisen, etwa im Rahmen von Personalgesprächen, Assessment-Centers oder Zielvereinbarungen (Busch 2000; Schlüter 2012).

Allen diesen begrifflichen Verwendungen des Terminus *Feedback* ist gemeinsam, dass sie auf eine Form der *Rückmeldung* Bezug nehmen, die auf eine vorhergehende Handlung oder Aktion folgt und Informationen bereitstellt. Die zugrunde liegende Intention und Funktion variiert jedoch erheblich: Während in technischen Systemen die Regulation im Vordergrund steht, betont die Verwendung im betrieblichen und organisationsbezogenen Kontext eher eine soziale und bewer-

tende Dimension. Angesichts der vielfältigen Verwendungsweisen erscheint eine kontextübergreifende Definition von *Feedback* kaum möglich und macht eine präzise begriffliche Einordnung des Terminus im Rahmen dieser Arbeit erforderlich.

Ditton (2014) weisen darauf hin, dass in vielen Publikationen auf solch eine explizite Begriffsbestimmung verzichtet wird, was zunächst auf einen vermeintlichen Konsens im Verständnis von Feedback hindeutet. Bei näherer Betrachtung zeigt sich jedoch, dass auch innerhalb des Bildungsbereichs vielfältige Bedeutungsnuancen mit dem Begriff verbunden sind. Um dieser begrifflichen Unschärfe zu begegnen, diskutiert Ditton (2014) eine systematische Abgrenzung der Begriffe *Feedback*, *Rückmeldung* und *Beurteilung*, welche im Folgenden aufgegriffen werden soll.

Feedback als intentionale Rückmeldung

In der instruktionspsychologischen Literatur zeigt sich ein funktionales Verständnis von Feedback, das insbesondere die zugrunde liegende Intention in den Blick nimmt. Im Zentrum steht die Frage, *ob* und *wie* eine Rückmeldung auf die gezielte Unterstützung von Lernprozessen ausgerichtet ist. Ein Blickwinkel, der sich schon bei Ramaprasad findet, welcher bereits 1983 postuliert, dass Informationen erst dann als Feedback gelten, wenn sie genutzt werden, um eine bestehende Lücke zu schließen (Ramaprasad 1983). Diese intentionale Bedingung findet sich auch in zahlreichen weiteren Definitionen im Bildungskontext wieder. So beschreiben Shute (2008) Feedback als eine *intentionale Kommunikationsform, die das Denken oder Verhalten von Lernenden mit dem Ziel verändern soll, Lernen zu verbessern.* Ähnlich argumentieren Boud und Molloy (2013), die Feedback als *Prozess* verstehen, bei dem Lernende Informationen über ihre Leistung erhalten – mit dem Ziel Unterschiede zwischen dem Ist-Zustand und einem angestrebten Soll-Zustand zu erkennen, um daraus Verbesserungen abzuleiten.

Im Rahmen dieser Dissertation wird die Definition von Feedback nach Narciss (2007) verwendet, die sowohl den informativen als auch den regulativen Charakter von Feedback betont:

Feedback

Feedback umfasst alle Informationen, die einer Person nach dem oder auch beim Bearbeiten von Lernaufgaben über ihren aktuellen Lern- oder/und Leistungsstand angeboten werden, mit dem Ziel, dass diese Informationen für die Regulation des Lernprozesses in Richtung erwünschter Ziele genutzt werden. (Narciss 2020)

Die Wahl dieser Definition folgt mehreren Gründen, die im Kontext dieser Dissertation zentral sind: Erstens verbindet sie – anders als viele eng gefasste Ansätze – den informativen und den regulativen Charakter von Feedback und stellt damit einen direkten Bezug zu Lern- und Steuerungsprozessen her. Diese Doppelfunktion ist insbesondere für digitale Lernumgebungen relevant, in denen Feedback nicht nur informieren, sondern auch das weitere Vorgehen der Lernenden anleiten soll. Zweitens ist die Definition in der instruktionspsychologischen Forschung breit rezipiert und bildet die theoretische Grundlage des *Interactive Two Feedback Loops*-Modells (Narciss 2020), das im weiteren Verlauf vor allem als konzeptionelle Grundlage für die technische Umsetzung von Feedbackprozessen dient (vgl. Abschnitt 2.1.5). Drittens bietet sie Anschluss an die Zielsetzung dieser Arbeit, weil sie Feedback als regulativen Prozess versteht, in dem Diskrepanzen zwischen aktuellem und gewünschtem Lernstand eine zentrale, jedoch nicht ausschließliche, Bezugsgröße darstellen. Damit bildet die Definition eine theoretisch und konzeptionell tragfähige Grundlage für die empirische Analyse und die technische Formalisierung kontextsensitiver Feedbackprozesse.

Abgrenzung zu Beurteilung

Der Begriff *Feedback* ist dabei vom Begriff *Beurteilung* abzugrenzen, der insbesondere im wirtschaftlichen wie auch im pädagogischen Kontext eine andere Funktion erfüllt. Während Feedback auf die Förderung von Lern- und Entwicklungsprozessen abzielt, dient Beurteilung primär der Bewertung von Leistungen, Kompetenzen oder Verhaltensweisen – meist mit selektiver, klassifizierender oder entscheidungsleitender Funktion. Eine strikte Trennung beider Konzepte ist in der Praxis jedoch kaum möglich und wird in der Fachliteratur kontrovers diskutiert, da die Übergänge oft fließend sind und Rückmeldungen beide Facetten enthalten können (Ditton 2014). In selektiven Verfahren wie Assessment-Centern kann z. B. zusätzlich entwicklungsorientiertes Feedback gegeben werden, während formative Rückmeldungen nicht selten implizit bewertende Elemente enthalten. Zudem basiert Feedback oft auf einer vorgelagerten Beurteilung, die jedoch nicht immer explizit ausgewiesen wird. Aus prozeduraler Perspektive lassen sich *Beurteilung* und *Feedback* daher als aufeinanderfolgende Komponenten eines gemeinsamen Verfahrenssystems verstehen – mit jeweils eigenen Zielen, Methoden und Wirkmechanismen (Bracken, Timmreck und Church 2001).

Im Rahmen dieser Arbeit wird – in Anlehnung an die in der Literatur diskutierte begriffliche Unschärfe zwischen Feedback, Beurteilung und Rückmeldung (vgl. Ditton (2014)) – eine differenzierte Verwendung dieser Begriffe angestrebt:

- **Rückmeldung** – als übergeordneter Begriff für jede Form von Reaktion auf eine Leistung oder Handlung.
- **Beurteilung** – als Rückmeldung mit klassifizierender oder selektierender Funktion, z. B. im Rahmen von Leistungsbewertungen.
- **Feedback** – als intentionale Rückmeldung mit dem Ziel, Lernprozesse zu regulieren und individuelle Entwicklung zu fördern.

2.1.2 Dimensionen und Formen von Feedback

Die Intention von Feedback, Lernprozesse zu regulieren und Lernende bei der Überwindung von Diskrepanzen zwischen aktuellem und angestrebtem Kompetenzstand zu unterstützen (Narciss 2020), kann auf viele unterschiedliche Weisen erfolgen. Denn Feedback ist nicht als einheitliches oder monolithisches Konstrukt zu verstehen, sondern umfasst eine Vielzahl von Erscheinungsformen, deren Wirkungspotenziale je nach Zielsetzung, Inhalt, Quelle, Zeitpunkt oder Darbietungsform stark variieren können (Ditton 2014). Lernprozesse selbst sind komplexe, dynamische Systeme, die durch individuelle, soziale und kontextuelle Faktoren geprägt werden (Hilpert und Marchand 2018) – entsprechend vielfältig sind auch die möglichen Ansätze, Feedback wirksam zu gestalten und einzusetzen.

Um die Vielfalt möglicher Einflussfaktoren theoretisch zu erfassen und didaktisch nutzbar zu machen, bietet die Fachliteratur verschiedene Klassifikationen und Typologien. Im Folgenden wird ein Bezugsrahmen vorgeschlagen, der sich an den klassischen W-Fragen orientiert und eine systematische wie differenzierte Betrachtung von Feedbackprozessen ermöglicht: *WER* gibt das Feedback (Quelle), *WAS* ist sein Gegenstand (Inhalt), *WIE* wird es vermittelt (Form), *WANN* erfolgt es (Timing) und *WARUM* wird es gegeben (Ziel/Funktion).

Es sei jedoch angemerkt, dass die genannten Dimensionen nicht immer trennscharf sind und sich somit teilweise überschneiden können. Der Überblick erhebt außerdem keinen Anspruch auf Vollständigkeit, sondern soll eine strukturierende Grundlage und begriffliche Orientierung für die weitere Ausführungen im Rahmen der Dissertation bieten.

2.1.2.1 Feedback-Quelle (WER)

Feedback lässt sich zunächst hinsichtlich seiner Quelle unterscheiden, also *wer* bzw. *was* das Feedback bereitstellt. Grundlegend wird dabei in der Literatur zwischen *internem* und *externem Feedback* differenziert. **Internes Feedback** entsteht durch die lernende Person selbst, etwa durch Selbstreflexion, Selbstevaluation oder metakognitive Prozesse. Lernende werten hier die Ergebnisse des eigenen Handelns aus und geben sich selbst Rückmeldung (Jacobs 2002). **Externes Feedback** wird hingegen vom Umfeld der Lernenden bereitgestellt, z. B. von Lehrkräften, Peers, Tutorinnen und Tutoren, technischen Systemen oder aus Reaktionen der Umwelt (Narciss 2012; Wichmann et al. 2014).

Draper (2005) differenziert diese Zweiteilung weiter und unterscheidet insgesamt drei Quellen von Feedback: (1) *Selbst-Feedback*, (2) Feedback aus der *Umwelt* (z. B. Aufgabenrückmeldungen, Performanz-Indikatoren) sowie (3) von einer *Lehrkraft* vermitteltes Feedback. Während (1) eindeutig dem *internen Feedback* zugeordnet werden kann, gehören (2) und (3) in die Kategorie des *externen Feedbacks*. Allerdings erweist sich insbesondere der Begriff der „Umwelt" als problematisch weit gefasst. Draper subsumiert darunter auch Fehlermeldungen oder Debugger-Ausgaben in Programmierumgebungen. Solche Informationen stellen zwar eine Form der Rückmeldung dar, verfolgen jedoch in erster Linie das Ziel, eine Fehleranalyse und -behebung zu unterstützen, nicht aber explizit den Lernprozess zu fördern. Die automatisierte Bereitstellung solcher Fehlermeldungen ohne didaktische Intention ist daher nicht als Feedback im engeren Sinne zu verstehen – obgleich sie durchaus Auslöser für *internes Feedback* sein kann, wenn Lernende die Informationen zur Selbstreflexion nutzen und ihre Strategie entsprechend anpassen. Es ist somit zu unterscheiden zwischen *automatisierter Rückmeldung* und *automatisiertem Feedback*. Von Letzterem soll im Rahmen der Dissertation nur dann die Rede sein, wenn die von technischen Systemen generierten Rückmeldungen mit der expliziten Zielsetzung erfolgen, eine menschliche Lehrperson in ihrer pädagogischen Funktion zu simulieren und dabei auf Abweichungen zwischen aktuellem Lernstand und angestrebtem Kompetenzniveau Bezug nehmen.

Im Zuge der Digitalisierung von Lehrprozessen ist die automatisierte Bereitstellung von *Rückmeldungen* zunehmend verbreitet. Für die Kategorisierung nach der Quelle erfordert dies jedoch eine weitergehende Differenzierung: Handelt es sich um automatisiertes Feedback, das auf Grundlage didaktischer Vorgaben regelbasiert bereitgestellt wird (z. B. korrektive Hinweise in E-Learning-Systemen), so kann die Quelle letztlich der Lehrkraft zugeschrieben werden, die über das System vermittelnd agiert. Wird automatisiertes Feedback hingegen von datengetriebenen, generativen Systemen wie großen Sprachmodellen bereitgestellt, ist die Quelle eher der Umwelt zuzuschreiben. In diesem Fall stammt die Rückmeldung aus Mustern großer Datenmengen und spiegelt keine spezifische pädagogische Intention wider.

Diese Unterscheidung macht deutlich, dass die Bestimmung der Feedback-Quelle im digitalen Kontext einer erweiterten Reflexion bedarf. Die Frage „Wer oder was gibt Feedback?" lässt sich nicht allein durch eine formale Zuweisung beantworten. Vielmehr muss der Blick auf die *Quelle* von Feedback aus zwei unterschiedlichen Blickwinkeln betrachtet werden:

1. **Quelle im Sinne der Entstehung:** Wurde das Feedback z. B. von einer Lehrkraft entwickelt, durch ein regelbasiertes System auf Basis einer didaktischen Modellierung erzeugt? Oder wurde es von einem technischen System (z. B. einem großen Sprachmodell) auf Grundlage großer Datenmengen generiert?
2. **Quelle im Sinne der Übermittlung:** Von wem erhalten die Lernenden das Feedback konkret? Wird es von einer Lehrkraft, durch Peers oder von einem technischen System übermittelt bzw. präsentiert?

Dass der Absender bzw. die Absenderin der Übermittlung als eigenständiger Wirkfaktor für Rückmeldungen gilt, ist ein Befund, der in verschiedenen Disziplinen als *Messenger-Effekt* (auch: *Messenger Bias*) beschrieben wird (Hafner, Elmes und Read 2019; Menon und Blount 2003). Studien zeigen, dass die Annahme und Umsetzung identischer Botschaften variieren kann, je nachdem, *wer* sie überbringt, *unabhängig* von ihrer ursprünglichen Entstehung (Hafner, Elmes und Read 2019; Maclean, Buckell und Marti 2019; Chou und Chen 2025). Für Bildungsprozesse legt dies nahe, die Gestaltung von Feedback nicht nur inhaltlich (Korrektheit, Spezifität, Elaborationsgrad), sondern auch *quellenbezogen* (Rolle, Expertise, Beziehung, wahrgenommene Agency) zu modellieren – insbesondere, wenn algorithmisch erzeugte Rückmeldungen von Lehrpersonen gerahmt, moderiert oder selektiv freigegeben werden.

Die Verteilung der genutzten Feedback-Quellen ist zudem *zeitlich dynamisch*. Vor der breiten Verfügbarkeit großer Sprachmodelle (LLMs) fungierten in vielen Kursen Lehrpersonen, Tutorinnen und Tutoren, Peer-Gruppen sowie aufgabenseitige Rückmeldungen (z. B. Tests, Compiler- oder IDE-Hinweise) als primäre Quellen. Lohr und Berges (2021) zeigen exemplarisch für dieses Prä-LLM-Setting, dass Studierende insbesondere Rückmeldungen von Lehrpersonen und automatisierten Bewertungssystemen als zentrale Informationsquellen für die Bearbeitung wöchentlicher Programmieraufgaben nutzen, wobei die Qualität und Nützlichkeit stark von der Aufgabenstruktur und der Art der bereitgestellten Hinweise abhängt. Mit dem Aufkommen generativer Systeme hat sich dieses verfügbare Quellenensemble jedoch deutlich erweitert: Lernende können nun *on demand* Rückmeldungen abrufen, deren *Entstehung* (datengetrieben, probabilistisch) und *Übermittlung* (Systemausgabe, ggf. von Lehrenden gerahmt) qualitativ von tradierten Quellen abweichen. Eine analoge Untersuchung wie die von Lohr und Berges (2021) in Post-LLM-Settings würde vermutlich ein deutlich *verschobenes Quellenprofil* zeigen – mit einer erheblich stärkeren Einbindung generativer Systeme als Feedbackquelle, wie es erste Untersuchungen bereits nahelegen (Keuning et al. 2024).

Schließlich ist Feedback in der Praxis selten monolithisch, sondern entsteht häufig aus *mehreren Quellen*: Rückmeldungen von Lehrenden, Peers, der lernenden Person selbst (internes Feedback) und technischen Systemen können kombiniert werden. In der Organisationspsychologie ist dies als *Multi-Source-Feedback* etabliert, primär zur mehrperspektivischen Leistungsbeurteilung (Ditton 2014). Übertragen auf Lehr-Lern-Kontexte eröffnet eine *didaktisch kuratierte* Kombination der Quellen neue Gestaltungsspielräume: Regelbasierte, erklärbare Komponenten können zuverlässig klassifizieren (z. B. Fehler-/Antwortklassen), während generative Komponenten elaborierte sprachliche Ausgestaltungen liefern. Lehrpersonen übernehmen die *Rahmung* (Aufgaben-/Kontextsetzung, Qualitätskontrolle), und Peers bzw. Lernende ergänzen reflektierende Perspektiven. Für diese Arbeit folgt daraus die Leitlinie, *Quelle als Gestaltungsvariable* explizit zu modellieren und in der Systemarchitektur so zu verankern, dass Entstehung und Übermittlung transparent gehalten, didaktisch steuerbar kombiniert und getrennt evaluiert werden.

Dass die *Quelle* von Feedback die wahrgenommene Glaubwürdigkeit, Relevanz und Wirksamkeit beeinflusst, ist empirisch belegt (Wichmann et al. 2014). Vor dem Hintergrund der oben eingeführten Unterscheidung zwischen *Entstehung* (wer generiert Inhalte?) und *Übermittlung* (wer präsentiert Inhalte?) bleibt jedoch weitgehend offen, wie Lernende Feedback bewerten, wenn beide Dimensionen *transparent* gemacht werden – etwa im Fall algorithmisch erzeugter, aber von einer Lehrperson kuratierter Rückmeldungen gegenüber menschlich verfasstem, jedoch systemvermitteltem Feedback. Die bewusste Wahrnehmung dieser doppelten Quellenattribution könnte somit nicht nur die Einschätzung der Glaubwürdigkeit, sondern auch das Ausmaß beeinflussen, in dem Lernende Feedback annehmen und in ihr Handeln integrieren.

2.1.2.2 Feedback-Inhalt (WAS)

Nachdem im vorherigen Abschnitt die Frage nach der *Quelle* von Feedback betrachtet wurde, rückt nun der inhaltliche Aspekt in den Fokus: WAS ist Gegenstand des Feedbacks, bzw. *welche Art der Informationen* sind im Feedback enthalten? Während sich die Dimension der Feedback-Quelle vergleichsweise klar strukturieren lässt, zeigt sich beim Feedback-Inhalt in der wissenschaftlichen Literatur eine außerordentliche Vielfalt. Kaum eine andere Dimension weist eine derart breite Palette an Klassifikationen und Typologien auf.
Dies hängt unmittelbar damit zusammen, dass Feedback inhaltlich sehr unterschiedlich ausgerichtet sein kann: Es reicht von einer bloßen Korrektheitsangabe über gezielte Hinweise zur Problemlösung bis hin zu metakognitiven Impulsen oder motivationalen Kommentaren. Entsprechend existieren zahlreiche Ansätze, die versuchen, Feedbackvarianten inhaltlich zu strukturieren und voneinander abzugrenzen. Die inhaltliche Dimension wird dabei in vielen empirischen Studien als die *wirksamste* Ebene der Differenzierung hervorgehoben (Shute 2008). In der Breite der möglichen Feedback-Inhalte finden sich die am häufigsten untersuchten und diskutierten Varianten von Feedback, die entscheidend zur Erklärung von Wirkungsunterschieden beitragen. Daher liegt der Schwerpunkt dieses Kapitels – und auch der weiteren Arbeit – insbesondere auf der inhaltlichen Dimension von Feedback.

Ein etablierter Klassifikationsansatz stammt von Narciss (2006), welche auf früheren Ordnungsversuchen u. a. von Kulhavy und Stock (1989) und Bangert-Drowns et al. (1991) aufbauend, eine Differenzierung in *einfache* und *elaborierte* Feedback-Typen vorschlägt:

Einfache Feedback-Typen (engl: **Simple Feedback**) enthalten Informationen, die sich auf grundlegende Informationen zur Korrektheit der Antwort oder Leistung beziehen, ohne weiterführende Erläuterungen oder Hinweise zu geben. Hier werden insgesamt drei Subtypen unterschieden:

Knowledge of Result/Response (`KR`) enthält lediglich Mitteilungen, ob eine gegebene Antwort oder Handlung richtig oder falsch war – ohne Angabe der korrekten Lösung oder weiterer Erläuterungen.

Knowledge of Correct Result (`KCR`) gibt eine korrekte Lösung an oder markiert diese.

Knowledge of Performance (`KP`) hingegen enthält Informationen über die Gesamtleistung oder des Leistungsstandes – beispielsweise durch Punkte, Prozentwerte oder Einstufungen im Vergleich zu anderen Lernenden.

Im Lichte der zuvor diskutierten begrifflichen Abgrenzung von *Rückmeldung*, *Beurteilung* und *Feedback* (vgl. Abschnitt 2.1.1) erscheint es zunächst naheliegend, *einfaches Feedback* nicht als Feedback im eigentlichen Sinne, sondern vielmehr als bloße *Rückmeldung* zu betrachten – insbesondere, da diese Formen ohne weiterführende Erläuterungen oder Hinweise auskommen. Bei näherer Betrachtung zeigt sich jedoch, dass auch diese einfacheren Formen von Feedback die in Abschnitt 2.1.1 beschriebene intentionale Funktion erfüllen können, sofern sie dazu dienen, Lernhandlungen gezielt zu steuern oder zu bestätigen.

Obwohl einfache Feedbackformen in der Literatur häufig als weniger wirksam beschrieben werden – teils sogar als „schlechtes Feedback" –, können sie unter bestimmten Bedingungen durchaus lernförderlich sein: Verfügen Lernende etwa über das notwendige Vorwissen, um aus einer Korrektheitsinformation eigenständig Schlussfolgerungen zu ziehen, kann die Bereitstellung zusätzlicher Erläuterungen nicht nur überflüssig, sondern sogar hinderlich sein, da sie selbstregulative Prozesse unterbrechen kann. So zeigen Bayerlein (2010) in einer empirischen Untersuchung, dass

insbesondere in klar strukturierten Aufgabenformaten und bei fortgeschrittenen Lernenden, einfache Feedback-Typen gezielt genutzt und als hilfreich wahrgenommen werden, während elaboriertere Formen häufig unbeachtet bleiben. Die Wirksamkeit einfachen Feedbacks hängt somit weniger von seiner elaborativen Tiefe als vielmehr von seiner Passung zur Lernsituation und den Vorkenntnissen der Lernenden ab.

Elaborierte Feedback-Typen (engl: **Elaborated Feedback**) hingegen gehen über die reine Information zur Korrektheit hinaus und fokussieren auf differenzierte inhaltliche Hinweise zur Unterstützung des Lernprozesses. Empirische Befunde belegen, dass elaboriertes Feedback durch seine erklärenden und instruktiven Zusatzinformationen tiefere kognitive Verarbeitungsprozesse anregt und somit ein hohes didaktisches Potenzial für individualisierte Lernunterstützung entfaltet (Wang et al. 2019; Wagner et al. 2024). Narciss (2006) unterscheidet fünf Subtypen elaborierten Feedbacks, die unterschiedliche Aspekte adressieren:

Knowledge about Task Constraints (`KTC`) bezieht sich auf Informationen über spezifische Anforderungen oder Rahmenbedingungen der Aufgabe, wie formale Vorgaben, Bearbeitungsregeln oder zulässige Lösungswege, ist also eng mit dem Aufgabenkontext verknüpft.

Knowledge about Concepts (`KC`) umfasst Informationen zu fachlichen Konzepten, Prinzipien oder Regeln, die für das Verständnis oder die Lösung der Aufgabe relevant sind. Diese Art von Feedback ist auf das konzeptuelle Lernen ausgerichtet und kann u. a. eine inhaltliche Vertiefung zum Ziel haben.

Mit *Knowledge about Mistakes* (`KM`) werden Fehler, die bei der Bearbeitung aufgetreten sind, gezielt thematisiert und erläutert. Solche diagnostischen Rückmeldungen können sowohl die Art als auch die mögliche Ursache eines Fehlers thematisieren und ermöglichen so eine differenzierte Fehleranalyse.

Knowledge on How to Proceed (`KH`), auch als *Know-How Feedback* bezeichnet, beinhaltet Hinweise zum weiteren Vorgehen. Hier können strategische oder prozedurale Empfehlungen enthalten sein, etwa zur Wahl eines geeigneteren Lösungswegs oder zur Strukturierung des nächsten Bearbeitungsschritts.

Knowledge about Metacognition (`KMC`) adressiert explizit die Ebene der Selbstregulation. Feedback dieser Art regt u. a. zur Reflexion des eigenen Lernprozesses an oder fordert zu einer Einschätzung des eigenen Verständnisses auf z. B. durch den Einsatz metakognitiver Leitfragen.

Neben der Unterscheidung zwischen *einfachem* und *elaboriertem* Feedback existieren in der Literatur weitere Klassifikationen, die unterschiedliche inhaltliche Aspekte fokussieren und sich zum Teil mit den von Narciss (2006) vorgeschlagenen Typen überschneiden:

Eine verbreitete Differenzierung erfolgt zwischen **konzeptuellem Feedback** und **prozeduralem Feedback**: Während ersteres auf das Verständnis grundlegender fachlicher Zusammenhänge und Konzepte abzielt – etwa durch Hinweise zu Definitionen, Prinzipien oder Zusammenhängen – bezieht sich prozedurales Feedback auf das korrekte Ausführen von Handlungsschritten oder Problemlösungsstrategien (Stovner und Klette 2022). Diese Unterscheidung deckt sich weitgehend mit den Typen *Knowledge about Concepts* (`KC`) und *Knowledge on How to Proceed* (`KH`) in der Typologie von Narciss. Besonders im Kontext des Programmierenlernens sind beide Formen bedeutsam: Konzeptuelles Feedback unterstützt z. B. beim Verständnis von grundlegenden Konzepten wie Kontrollstrukturen, Datentypen oder Funktionen, während prozedurales Feedback Hinweise zu konkreten Implementierungsansätzen, Code-Struktur oder Debugging geben kann.

Eine weitere bekannte Klassifikation betrifft die Unterscheidung zwischen **evaluativem** und **informativem** bzw. **tutoriellem Feedback** (Maier und Klotz 2022):
Evaluatives Feedback beschränkt sich in der Regel auf die Bewertung oder Einstufung einer Leistung – etwa durch die Angabe von Punkten, Noten oder qualitativen Urteilen wie „gut" oder „unzureichend". Es entspricht in seiner Funktion primär *Knowledge of Performance* (`KP`) oder in Teilen *Knowledge of Result* (`KR`). Demgegenüber liefert *informatives* bzw. *tutorielles Feedback* weiterführende inhaltliche Hinweise, die Lernende bei der Überarbeitung ihrer Leistung oder beim Verständnis ihrer Fehler unterstützen sollen (Narciss 2006). Es zielt nicht auf Bewertung, sondern auf Förderung ab, und umfasst damit insbesondere die elaborierten Feedbackformen wie `KC`, `KM` oder `KH`. Narciss verwendet den Begriff *informatives tutorielles Feedback* explizit, um solche lernunterstützenden Rückmeldungen abzugrenzen.

Bei **ergebnisorientiertem Feedback** (Paulson Gjerde, Padgett und Skinner 2017) handelt es sich hingegen um Rückmeldungen, die sich ausschließlich auf das *Endprodukt* oder das *Lösungsergebnis* beziehen – unabhängig vom Lösungsweg. Dieses Feedback fokussiert also auf das Artefakt eines Lernergebnisses wie z. B. den konkreten Quellcode, nicht aber den *Prozess*, wie dieser entstanden ist. Entsprechend fällt es in die Kategorie `KR` oder `KCR` und wird häufig in standardisierten Test- oder Prüfungskontexten eingesetzt. Im Gegensatz dazu fokussiert das zuvor genannte **prozedurale Feedback** auf die Analyse und Förderung des Lösungswegs und wird für formative Settings als wirksamer eingeschätzt (Shute 2008).

Speziell für den Bereich des computerbasierten Lernens wurde von Narciss und Huth (2006) der Begriff des **bug-related tutoring feedback** eingeführt. Hierunter fallen Rückmeldungen, die spezifisch auf das Auffinden, Erklären und Beheben von Fehlern abzielen. Diese Art von Feedback greift häufig auf Elemente von `KM`-Feedback zurück und kann mit Aspekten von `KH`-Feedback kombiniert werden, wenn zusätzlich Hinweise zur Korrektur oder zur Vermeidung ähnlicher Fehler gegeben werden. Die besondere Herausforderung besteht dabei darin, nicht nur den Fehler zu identifizieren, sondern diesen auch verständlich zu erklären und gegebenenfalls den zugrunde liegenden konzeptuellen Irrtum (`KC`) aufzudecken (Keuning, Jeuring und Heeren 2019).

Aufbauend auf der Taxonomie von Narciss (2006) greifen Keuning, Jeuring und Heeren (2019) und Keuning (2020) diese Perspektive auf und erweitern sie für den Kontext automatisierter Programmierumgebungen. Die Hauptkategorien bleiben dabei erhalten, werden jedoch stärker auf die spezifischen Anforderungen des Programmierlernens bezogen und um domänenspezifische Subtypen ergänzt. Tabelle 2.1 gibt einen Überblick über die Feedback-Typen mitsamt den jeweiligen Subtypen und kurzen Beschreibungen. Die Taxonomie bildet zugleich die Grundlage für die in dieser Arbeit vorgenommene Analyse von Feedback in Programmierkontexten. Offen bleibt allerdings, inwieweit die dort entwickelten Kategorien auch für durch Lehrpersonen bereitgestelltes Feedback in Programmierkontexten geeignet sind (vgl. Kapitel 4).

Insgesamt verdeutlichen die dargestellten Klassifikationen, dass der Inhalt von Feedback hochgradig variabel ist und unterschiedliche kognitive Ebenen adressieren kann. Für die Analyse von Feedback in Lernsystemen – insbesondere im Programmierkontext – bietet die Kombination dieser Perspektiven ein reiches Vokabular zur Beschreibung, Bewertung und Gestaltung von Feedback.

2.1.2.3 Feedback-Form (WIE)

In computerbasierten Lernumgebungen eröffnen moderne Technologien eine Vielzahl an Möglichkeiten der Feedbackpräsentation – von einfachen Textanzeigen bis hin zu multimodalen, interaktiven Rückmeldungen mit animierten Agenten oder gesprochener Sprache. Die Art WIE Feedback präsentiert wird kann ebenfalls Auswirkung auf dessen Wirksamkeit haben (Panadero und Lipnevich 2022). Damit sind sowohl die medial-technische Darbietung als auch die Struktu-

Kat.	Bezeichnung	Definition
KTC	**Knowledge about task constraints**	Feedback zu den Anforderungen und Regeln einer Aufgabe.
TR	Hints on task requirements	Hinweise zu spezifischen Anforderungen der Aufgabe (z. B. notwendige Sprachelemente, verbotene Methoden).
TPR	Hints on task-processing rules	Hinweise auf allgemeine Vorgehensweisen oder eingebaute Bearbeitungsregeln der Aufgabe (z. B. Schritt-für-Schritt-Strategien, wie bei rekursiven Aufgaben vorzugehen ist).
KC	**Knowledge about concepts**	Feedback zu fachlichen Konzepten und Inhalten der Aufgabe.
EXP	Explanations on subject matter	Erklärungen zu relevanten Konzepten oder Sprachkonstrukten.
EXA	Examples illustrating concepts	Beispiele oder Analogien, die fachliche Konzepte veranschaulichen.
KM	**Knowledge about mistakes**	Feedback zu Fehlerarten in der Lösung.
TF	Test failures	Rückmeldungen zu fehlschlagenden Testfällen bzw. Testergebnissen.
CE	Compiler errors	Rückmeldungen zu syntaktischen Fehlern, die vom Compiler/Interpreter erkannt werden.
SE	Solution errors	Hinweise auf semantische oder logische Fehler in der Lösung.
SI	Style issues	Rückmeldungen zu Stil- und Layoutproblemen im Code (z. B. Benennung, Formatierung).
PI	Performance issues	Hinweise auf Effizienz- oder Laufzeitprobleme der Lösung.
KH	**Knowledge about how to proceed**	Feedback, das beim Weiterarbeiten oder Verbessern unterstützt.
EC	Bug-related hints for error correction	Hinweise, wie konkrete Bugs (Syntax- oder Semantikfehler) zu beheben sind.
TPS	Task-processing steps	Hinweise zu nächsten Bearbeitungsschritten, um bei der Aufgabenlösung voranzukommen.
IM	Improvement hints	Hinweise zur Verbesserung der Qualität des Programms (z. B. Stil, Effizienz).
KMC	**Knowledge about meta-cognition**	Feedback, das die Reflexion und Selbstregulation beim Lernen unterstützt.

Tabelle 2.1: Feedbacktypen nach Keuning (2020)

rierung und Sequenzierung der Informationsanteile gemeint. Narciss (2007) weist darauf hin, dass die Form der Rückmeldung nicht nur als technisches Merkmal zu verstehen ist, sondern didaktisch gezielt gestaltet werden sollte. Dabei lassen sich verschiedene Aspekte der Feedback-Form unterscheiden.

Feedback kann in unterschiedlicher Modalität dargeboten werden – etwa schriftlich, mündlich, visuell (z. B. als Diagramm) oder auditiv (z. B. als Sprachausgabe). Während **unimodales Feedback** sich auf eine dieser Modalitäten beschränkt, kombiniert **multimodales Feedback** mehrere

Präsentationsformen. Letzteres wird insbesondere in multimedialen Lernumgebungen eingesetzt, wobei die Prinzipien der kognitiven Belastung und der Multimedia-Lerntheorie beachtet werden sollten (Mayer 2001).

Gerade im Bereich des Programmierlernens ist der Zusammenhang zwischen Feedbackform und kognitiver Belastung besonders relevant. Nach der *Cognitive Load Theory* von Sweller (1988) und Sweller, Van Merrienboer und Paas (1998) steht Lernleistung in engem Zusammenhang mit der begrenzten Verarbeitungskapazität des Arbeitsgedächtnisses. Lernende müssen gleichzeitig die Problemstruktur, das Feedback und die eigenen Lösungsstrategien verarbeiten. Wird Feedback zu umfangreich, redundant oder in mehreren, nicht aufeinander abgestimmten Modalitäten präsentiert, kann dies zu einer Erhöhung der *extraneous cognitive load* führen – also jener Belastung, die nicht der eigentlichen Aufgabenbearbeitung dient, sondern durch die Art der Informationsdarstellung entsteht. Dies kann insbesondere bei komplexen Programmieraufgaben, die selbst bereits hohe Anforderungen an das Arbeitsgedächtnis stellen, zu einer Beeinträchtigung des Lernprozesses führen.

Für die Gestaltung von Feedback beim Programmieren folgt daraus, dass Rückmeldungen in Umfang, Detaillierung und Modalität so gestaltet sein sollten, dass sie die *germane cognitive load* – also die lernrelevante Verarbeitung – unterstützen, ohne zusätzliche Belastung durch übermäßige Präsentationskomplexität zu erzeugen. Multimodale oder visuelle Darstellungen (z. B. Markierungen im Code, erklärende Diagramme oder Schritt-für-Schritt-Hinweise) können dabei lernförderlich sein, wenn sie komplementär und kohärent aufeinander abgestimmt sind. Werden sie hingegen unstrukturiert oder redundant eingesetzt, besteht die Gefahr einer kognitiven Überlastung und damit einer reduzierten Feedbackwirksamkeit (Mayer 2001; Sweller 1988; Narciss 2007; Sweller, Van Merrienboer und Paas 1998). Empirische Untersuchungen zur optimalen Gestaltung solcher multimodalen Rückmeldungen im Programmierkontext stehen bislang jedoch noch aus.

Feedback kann dabei entweder gleichzeitig (simultan) oder schrittweise (sequenziell) dargeboten werden. Bei der **simultanen Präsentation** werden alle Feedback-Elemente auf einmal angezeigt, was bei komplexen Rückmeldungen zu einer hohen kognitiven Belastung führen kann. Die **sequentielle Präsentation** hingegen strukturiert den Inhalt des Feedbacks in einzelne Informationsschritte, die nacheinander angeboten werden – etwa im Rahmen mehrerer Versuche oder über fortschreitende Hilfestufen. Studien deuten darauf hin, dass sequenzielle Formate tutoriellen Feedbacks insbesondere bei komplexen Aufgaben positive Effekte im Hinblick auf Lerneffektivität erzielen können (Narciss 2007).

Ein weiteres Unterscheidungsmerkmal liegt darin, ob Lernende nach Erhalt von Feedback eine erneute Bearbeitung der Aufgabe vornehmen können. **Multiple-Try-Feedback** erlaubt es, auf Basis der Rückmeldung mehrere Lösungsversuche durchzuführen. Solche Formate können eine aktive Nutzung von Feedback zur Selbstkorrektur fördern. Werden Feedbackkomponenten passend gestaltet, so kann das Feedback Lernende dazu anregen eigene Korrekturen vorzunehmen, bevor die korrekte Lösung offengelegt wird (Narciss 2007).

Im Gegensatz zu **statischem Feedback**, das unabhängig von individuellen Lernvoraussetzungen oder situativen Faktoren in identischer Form bereitgestellt wird, passt sich **adaptives Feedback** dynamisch an Merkmale der jeweiligen Lernsituation und der Lernenden an. Dabei können Inhalt, Umfang oder Form der Rückmeldung in Abhängigkeit vom Vorwissen, von Reaktionen auf frühere Hinweise oder von der Einschätzung des Lernstands variiert werden (Le 2016). Adaptives Feedback ist insbesondere Bestandteil intelligenter Tutoring-Systeme (siehe Abschnitt 2.3.3) und erweist sich als lernwirksamer als standardisierte, nicht-adaptive Rückmeldungen, sofern die Anpassung auf validen Diagnosen beruht (Roll et al. 2011).

Sales (1993) unterscheidet dabei zwischen *adaptiertem* und *adaptivem Feedback*. Ersteres bezieht sich auf Rückmeldungen, die *vorab* an bestimmte Bedingungen oder Merkmale der Lernenden angepasst werden. Diese basieren auf vordefinierten Regeln oder Kategorien, die durch vorherige Analysen festgelegt wurden, und werden anschließend an alle Lernenden in gleicher Weise ausgegeben. Adaptives Feedback hingegen orientiert sich an den individuellen Lernvoraussetzungen und Bedürfnissen und passt sich in Echtzeit an beobachtetes Verhalten oder Leistungsentwicklungen an. Dabei kann die Elaboriertheit des Feedbacks – etwa die Tiefe der Erklärung oder die Art der Unterstützung – zwischen erfolgreichen und weniger erfolgreichen Lernenden variieren (Sales 1993; Krause 2007).

Neuere Ansätze präzisieren das *Wie* adaptiver Feedbackprozesse durch eine systematische, datenbasierte Perspektive. Nach Bauer et al. (2025) erfolgt Adaptivität durch die fortlaufende Erfassung und Integration relevanter Lernmerkmale – darunter kognitive, metakognitive, motivational-affektive und soziale Indikatoren – in ein zugrunde liegendes Entscheidungsmodell. Das im sogenannten *SHARP-Framework* beschriebene Prinzip strukturiert diesen Anpassungsprozess entlang zentraler Fragen wie *Was* angepasst wird, *Wann* die Anpassung erfolgt und *Auf welcher Grundlage* sie ausgelöst wird. Damit wird deutlich, dass sich das *Wie* adaptiver Feedbacksteuerung nicht allein auf die Form der Rückmeldung bezieht, sondern auf die systematische Nutzung lernprozessbezogener Daten, um Rückmeldungen kontextsensitiv und individuell zugleich zu gestalten.

2.1.2.4 Feedback-Timing (WANN)

Neben der Quelle, dem Inhalt und der Form von Feedback spielt auch der *Zeitpunkt* bzw. die Frage WANN Feedback verfügbar sein soll, eine zentrale Rolle (Narciss 2020). Diese bezieht sich dabei jedoch nicht ausschließlich auf chronologische Zeit im engeren Sinne, sondern auf die *Position* des Feedbacks innerhalb des Lern- oder Bearbeitungsprozesses. Es geht also darum, zu welchem Zeitpunkt das Feedback in Relation zum Lernereignis verfügbar gemacht wird – sei es *vor*, *während* oder *nach* einer Handlung, Antwort oder Aufgabenbearbeitung (Candel et al. 2020). Die Forschungsliteratur unterscheidet dabei mehrere zeitliche Ordnungsprinzipien, die sich komplementär zueinander betrachten lassen:

Eine häufig untersuchte Unterscheidung betrifft die Differenz zwischen *sofortigem Feedback* und *verzögertem Feedback*. Bei **sofortigem Feedback** wird die Rückmeldung unmittelbar nach der Antwort oder Handlung präsentiert. Diese Form ist insbesondere in digitalen Lernumgebungen verbreitet und ermöglicht eine direkte Verknüpfung zwischen Aktion und Feedback, was die Fehlerlokalisierung und -korrektur erleichtern kann. In der behavioristischen Tradition wird diese Form häufig als besonders wirksam betrachtet, da sie eine direkte Verstärkung (im Sinne von *reinforcement*) ermöglicht (Kulik und Kulik 1988).

Bei **verzögertem Feedback** hingegen wird die Rückmeldung erst mit einem gewissen zeitlichen Abstand zum Lernereignis bereitgestellt – z. B. am Ende einer Aufgabe, einer Lerneinheit oder nach Abschluss des gesamten Lernprozesses. Studien zeigen, dass verzögertes Feedback Vorteile im Hinblick auf langfristige Behaltensleistungen bieten kann – ein Effekt, der als *delay-retention effect* bezeichnet wird (Kulhavy und Stock 1989; Brackbill et al. 1963). Insbesondere bei komplexen Aufgaben kann ein zeitlicher Abstand dazu beitragen, dass Lernende ihre Antworten vor dem Feedback selbst reflektieren, wodurch tiefere Verarbeitungsprozesse angeregt werden können (Mathan und Koedinger 2005). Welcher Zeitpunkt dabei lernförderlicher ist, hängt von verschiedenen Faktoren ab – u. a. von der Komplexität der Aufgabe, dem angestrebten Lernziel und den metakognitiven Fähigkeiten der Lernenden. Auch Narciss (2007) betont, dass die Effekte unterschiedlicher Feedback-Zeitpunkte im Kontext der jeweiligen Lernaufgabe sowie der Zielstellung (z. B. Fehlersensibilisierung, Transferförderung) bewertet werden müssen.

Neben dem konkreten Zeitpunkt für Feedback stellt sich auch die Frage, ob und wie Lernende die Verfügbarkeit von Feedback selbst steuern können. Eine grundlegende Unterscheidung existiert dabei zwischen der sog. *presearch availability* und der *kontrollierte Verfügbarkeit* (Shute 2008; Bangert-Drowns et al. 1991):

Bei **presearch availability** ist Feedback von Beginn an verfügbar – beispielsweise durch eingeblendete Lösungshinweise, einsehbare Bewertungsraster oder sofort sichtbare Fehlermarkierungen. Solche Formate können insbesondere bei offenen Lernumgebungen hilfreich sein, bergen jedoch auch die Gefahr, dass Lernende Feedback abrufen, ohne sich zuvor aktiv mit dem Problem auseinanderzusetzen.

Kontrollierte Verfügbarkeit hingegen bedeutet, dass Feedback erst nach bestimmten Aktionen, Bearbeitungsschritten oder auf Anforderung hin bereitgestellt wird – etwa nach einem vollständigen Lösungsversuch oder auf Knopfdruck. Diese Form wird häufig in intelligenten Tutoring-Systemen oder interaktiven Lernplattformen implementiert und zielt darauf ab, die selbstregulierte Nutzung von Feedback zu fördern (Aleven et al. 2006).

Ein Beispiel für eine solche bewusst gesteuerte Form des Feedbacks bietet das System `GATE` (*Generic Assessment & Testing Environment*) (Strickroth, Olivier und Pinkwart 2011). Hier erhalten Lernende nach der Abgabe ihrer Lösung Zugriff auf (ggf. limitierte) automatische Programmtests, die einzeln angefordert werden können. Jeder Test kann dabei nur in einer festgelegten Anzahl ausgeführt werden und gibt ausschließlich eine knappe Rückmeldung über Erfolg oder Misserfolg. Ziel dieser Gestaltung ist es, eine unreflektierte Nutzung von Hilfestellungen zu vermeiden und Lernende zu einer bewussten Auseinandersetzung mit ihren Lösungsansätzen zu motivieren. `GATE` stellt damit ein Beispiel für ein System dar, in dem Feedback sowohl in seiner *Verfügbarkeit* (auf Anforderung) als auch in seiner *Quantität* (limitierte Abrufe) bewusst begrenzt wird, um den selbstregulierten Umgang mit Feedback zu fördern.

Eine weitere Perspektive auf das Feedback-Timing bezieht sich auf dessen Positionierung im Lernverlauf. Die Literatur unterscheidet zwischen Feedback, welches VOR einem Ereignis (*proaktiv/prädiktiv*), WÄHREND des Ereignisses (*formativ*) oder NACH dem Ereignis (*summativ*) gegeben werden (Dainton 2018):

Proaktives Feedback erfolgt vor der Handlung und bereitet Lernende auf typische Fehler, relevante Strategien oder Anforderungen der Aufgabe vor. Es hat eine vorbereitende, aktivierende Funktion und wird beispielsweise in Form von Leitfragen, Warnhinweisen oder metakognitiven Hinweisen eingesetzt.

Formatives Feedback begleitet den Lernprozess und zielt auf die unmittelbare Steuerung und Regulation der Bearbeitung ab. Es setzt typischerweise an einer Teillösung oder einem Bearbeitungsschritt an und bietet zeitnahe Unterstützung, um Fehlentwicklungen frühzeitig zu korrigieren. Formatives Feedback ist insbesondere in adaptiven Lernsystemen oder tutorbasierten Lernumgebungen zentral.

Summatives Feedback hingegen erfolgt am Ende eines Lernprozesses und dient der Rückschau auf das Erreichte. Es wird häufig in Prüfungs- oder Testkontexten eingesetzt und informiert über den Grad der Zielerreichung. Dabei kann es sowohl evaluativen (z. B. Punktevergabe; siehe auch *evaluatives Feedback* bzw. `KP`-Feedback in Abschnitt 2.1.2.2) als auch informativen Charakter (z. B. Musterlösung, Stärken-Schwächen-Analyse; siehe auch *informatives Feedback* bzw. `KCR`-Feedback in Abschnitt 2.1.2.2) haben.

Die Beispiele verdeutlichten, dass Art und Zeitpunkt der Feedbackbereitstellung einen Einfluss darauf haben, wie Lernende Rückmeldungen nutzen. Werden diese sehr früh oder ohne eigene Anforderung zugänglich gemacht, besteht die Gefahr einer eher passiven und unreflektierten

Nutzung. Eine begrenzte Verfügbarkeit – wie sie im GATE-System umgesetzt ist – kann dagegen dazu beitragen, dass Lernende Rückmeldungen gezielter einsetzen und bewusster in ihre Überarbeitungsstrategien integrieren. Die Steuerung der Verfügbarkeit wirkt damit nicht nur auf die Informationsaufnahme, sondern auch auf motivationale und metakognitive Prozesse: Während ein zu leichter Zugriff die aktive Auseinandersetzung mit der Aufgabe mindern kann, birgt eine zu starke Einschränkung das Risiko, den Lernfluss zu unterbrechen oder Frustration hervorzurufen.

2.1.2.5 Feedback-Funktion (WARUM)

Neben den bisher genannten Dimensionen von Feedback ist schließlich auch die Frage nach seiner *Funktion* zentral: WARUM wird Feedback gegeben, bzw. welchen Zweck soll es im Lernprozess erfüllen?

Wie bereits in Abschnitt 2.1.1 diskutiert verfolgt Feedback grundlegend immer das Ziel, Lernprozesse zu unterstützen und zu fördern. Die konkrete Funktion kann jedoch je nach Kontext, Zielsetzung und Feedback-Typ variieren:

Narciss (2007) nennt drei übergeordnete Funktionen, die sich auf unterschiedliche psychologische Prozesse beziehen: eine **kognitive**, eine **motivationale** sowie eine **metakognitive Funktion**. Im folgenden Abschnitt werden diese drei Funktionen näher erläutert, wobei anzumerken ist, dass sie in der Praxis nicht isoliert zu betrachten sind, sondern in Wechselwirkung zueinander stehen können.

Die kognitive Funktion von Feedback zielt auf die Förderung des Wissenserwerbs, des Verständnisses und der Problemlösefähigkeiten ab. Sie steht im Mittelpunkt der meisten empirischen Untersuchungen zur Wirksamkeit von Feedback, das in diesem Sinne auf folgende Wirkmechanismen abzielen kann:

- *Fehlerrückmeldung:* Unterstützung bei der Identifikation und Korrektur von Fehlern, z. B. durch die Typen `KM` oder `KCR`.
- *Konzeptvermittelnde Funktion:* Erklärung von Zusammenhängen, Prinzipien oder Regeln, etwa durch `KC`.
- *Strategieunterstützung:* Förderung geeigneter Lösungsstrategien durch Hinweise zum weiteren Vorgehen (`KH`).
- *Anpassung mentaler Modelle:* Hilfestellung beim Umstrukturieren von Vorwissen oder dem Aufbau neuer kognitiver Strukturen.

Feedback kann auch eine bedeutende Rolle für Motivation, Selbstwirksamkeit und Zielorientierung spielen. Narciss (2007) nennt hier insbesondere:

- *Bestätigung oder Ermutigung:* Positive Rückmeldung kann das Vertrauen in die eigene Kompetenz stärken.
- *Förderung von Lernzielen:* Feedback kann dazu beitragen, leistungs- vs. lernzielorientiertes Verhalten zu steuern.
- *Aufmerksamkeitssteuerung:* Rückmeldungen können die Aufmerksamkeit gezielt auf relevante Aspekte der Aufgabe lenken.
- *Förderung von Anstrengungsbereitschaft:* Motivation kann gestärkt werden, z. B. durch Feedback, das auf Fortschritte oder verbleibende Entwicklungspotenziale verweist.

Diese Wirkungen hängen stark von der Art der Formulierung und dem wahrgenommenen „Ton“ des Feedbacks ab.

Schließlich kann Feedback auch metakognitive Prozesse anregen und so zur Selbstregulation des Lernens beitragen. Mögliche Funktionen sind hier:

- *Unterstützung bei der Selbsteinschätzung:* Feedback kann Lernenden dabei helfen, den eigenen Leistungsstand realistisch zu bewerten.
- *Anregung zur Reflexion:* Hinweise regen dazu an, über gewählte Lösungswege oder Denkstrategien nachzudenken (z. B. durch `KMC`).
- *Förderung von Planung und Monitoring:* Rückmeldungen unterstützen dabei, das eigene Lernen zu planen, zu überwachen und gegebenenfalls anzupassen.

Gerade im Kontext digitaler Lernumgebungen mit hoher Eigenverantwortung wird diese Funktion zunehmend als wichtig erachtet, etwa zur Förderung selbstregulierten Lernens.

2.1.3 Wirksamkeit von Feedback

Die vielfach betonte Bedeutung von Feedback in Bildungsprozessen wirft die zentrale Frage auf, unter welchen Bedingungen es tatsächlich lernwirksam ist. Zahlreiche Studien zeigen positive Effekte von Feedback auf den Lernfortschritt, gleichzeitig ist die empirische Befundlage keineswegs einheitlich. Die Wirkung von Feedback ist nicht per se gegeben, sondern hängt von einer Vielzahl von Einflussfaktoren ab – darunter die zuvor beschriebenen Dimensionen Feedback-Quelle, -Inhalt, -Form, -Timing und -Funktion sowie weitere situative und individuelle Merkmale der Lernenden. Entscheidend ist zudem, in welchem Verhältnis die Rückmeldung bewertet wird, also welche *Bezugsnorm* der Wirksamkeit zugrunde liegt – ob sie sich etwa auf den individuellen Lernfortschritt, auf soziale Vergleiche oder auf ein kriteriales Leistungsziel bezieht (Butler 1987; Kluger und DeNisi 1996).

Trotz hoher interner Validität vieler Experimentalstudien ist die Evidenzlage zur Wirksamkeit verschiedener Feedbackarten in weiten Teilen uneindeutig. So zeigen Meta-Analysen wie die von Kluger und DeNisi (1996), dass Feedback insgesamt zwar im Mittel positive Effekte auf Performanz hat, aber bis zu einem Drittel der untersuchten Studien auch negative Effekte fanden. Ein möglicher Grund für diese Inkonsistenzen liegt in der großen Heterogenität der untersuchten Settings und Feedbackbedingungen (Ditton 2014). Diese Komplexität erschwert die Generalisierbarkeit von Einzelergebnissen und verdeutlicht die Notwendigkeit kontextspezifischer, systematischer Untersuchungen.

Verschiedene Studien zeigen weiterhin, dass nicht alle Feedbackformen gleichermaßen lernwirksam sind (Kluger und DeNisi 1996; Hattie und Timperley 2007; Shute 2008; Narciss 2007). Insbesondere elaborierte, informationsreiche Rückmeldungen (z. B. `KM`-Feedback, `KH`-Feedback) zeigen häufig stärkere Effekte auf Verständnis und Problemlösefähigkeit als einfache Korrekturhinweise (z. B. `KR`-Feedback) oder bewertende Rückmeldungen (z. B. `KP`-Feedback) (Narciss 2007). Allerdings sind diese Ergebnisse ebenfalls nicht konsistent. In einigen Fällen zeigte elaboriertes Feedback keine signifikant bessere Wirkung als einfaches Feedback, etwa wenn kognitive Überlastung auftrat oder wenn die Rückmeldung nicht zum Vorwissen der Lernenden passte. Auch kann ein Zuviel an Information die Selbstregulation beeinträchtigen.

Ein weiterer zentraler Faktor ist der Zeitpunkt des Feedbacks. Studien zeigen, dass sofortiges Feedback in behavioristischen Settings oder bei niedrigschwelligen Aufgaben kurzfristige Performanz verbessern kann, während verzögertes Feedback positive Effekte auf langfristige Behaltensleistungen (*delay-retention effect*) und tiefere kognitive Verarbeitung haben kann (Kulhavy und Stock 1989; Brackbill et al. 1963).

Mathan und Koedinger (2005) zeigen, dass insbesondere bei komplexen Aufgaben ein zeitlicher Abstand helfen kann, eigene Denkfehler zu reflektieren und die Wirksamkeit des Feedbacks zu erhöhen. Gleichzeitig legen Studien wie (Candel et al. 2020) nahe, dass allein das Wissen über die Verfügbarkeit von Feedback das Lernverhalten beeinflusst – z. B. durch reduzierten initialen Aufwand in Erwartung einer zweiten Chance. Diese Effekte müssen in der Gestaltung von Feedback berücksichtigt werden.

Trotz dieser komplexen und teils widersprüchlichen Befundlage lassen sich einige Merkmale identifizieren, die in der Literatur übereinstimmend als lernförderlich angesehen werden:

Inhaltliche Qualität: Feedback sollte spezifisch, verständlich und aufgabenbezogen sein – vage oder generische Rückmeldungen sind oft wenig wirksam.

Zeitliche Passung: Das Timing sollte dem Lernziel angepasst sein – unmittelbare Rückmeldung bei Übungsaufgaben, verzögerte bei Transfer- oder Reflektionszielen.

Anpassung an Lernvoraussetzungen: Feedback sollte das Vorwissen, die kognitiven und metakognitiven Kompetenzen der Lernenden berücksichtigen
(vgl. adaptives Feedback in Abschnitt 2.1.2.3).

Selbstregulationsförderung: Rückmeldungen, die Reflexion anregen oder Lernstrategien adressieren, zeigen in vielen Studien nachhaltigere Effekte (Narciss 2007).

Diese Qualitätsmerkmale markieren gemeinsame Nenner der Befundlage, ersetzen jedoch keine kontextbezogene Ausgestaltung. Denn wie die Datenlage zeigt, hängt *wie* und *unter welchen Bedingungen* Feedback wirkt, wesentlich von Aufgabenkontext, Zielsetzung und Lernvoraussetzungen ab (Kluger und DeNisi 1996; Shute 2008).

Die Wirksamkeit von Feedback lässt sich daher nicht isoliert bewerten, sondern ist das Ergebnis eines komplexen Zusammenspiels verschiedener Einflussgrößen (Hattie und Timperley 2007). Dies erschwert direkte Vergleiche zwischen Studien und macht deutlich, dass es keine universelle „beste" Form von Feedback gibt.

Vor diesem Hintergrund erscheint es für diese Arbeit geboten, systematische Modelle zu entwickeln, die sowohl Aufgabenmerkmale, Feedbackdimensionen als auch lernseitige Voraussetzungen berücksichtigen. Die in Kapitel 5 dargestellte Modellierung von Aufgaben- und Kontextstrukturen zielt genau darauf ab: Sie bildet eine Grundlage für die Analyse und technische Umsetzung kontextsensitiven, qualitativ hochwertigen Feedbacks und stellt damit eine Grundlage für die kontextsensitive automatisierte Bereitstellung von Feedback sowie eine strukturierte und vergleichbare Erforschung dessen Wirksamkeit dar.

2.1.4 Feedback-Strategien als integratives Rahmenkonzept

Die bisherigen Abschnitte haben gezeigt, dass Feedback ein vielschichtiges, multidimensionales Konstrukt ist. Seine Wirksamkeit hängt nicht allein von einzelnen Dimensionen – etwa dem Inhalt, dem Zeitpunkt oder der Darbietungsform – ab, sondern von der Kombination und Passung dieser Dimensionen in Bezug auf individuelle und situative Merkmale. Daraus ergibt sich die Notwendigkeit, Feedbackprozesse nicht isoliert zu betrachten, sondern als Teil eines strategischen Gesamtkonzepts zu verstehen. In der Literatur wird dies durch den Begriff der *Feedback-Strategie* aufgegriffen (Narciss 2020). Eine Feedback-Strategie ist dabei als ein koordinierter Plan zu verstehen, der die verschiedenen Komponenten von Feedback – darunter Inhalt, Funktion, Timing, Präsentation und situative Bedingungen – aufeinander abstimmt, um lernförderliche Wirkungen zu erzielen. Sie spezifiziert damit nicht nur *was* für ein Feedback gegeben wird, sondern u. a. *wann*, *wie*, *warum* und *unter welchen Bedingungen* (siehe Dimensionen und Formen von Feedback in Abschnitt 2.1.2).

Narciss (2012) definiert eine Feedback-Strategie wie folgt:

Feedback-Strategie

Eine Feedback-Strategie ist ein koordinierter Plan, der klare und entschiedene Aussagen integriert. Darunter mindestens (a) unter welchen situativen und individuellen Bedingungen des Unterrichtskontexts (Feedback-Bedingungen), (b) welche externen Informationen nach der Reaktion bereitgestellt werden sollten (Feedback-Inhalt), (c) welche (Unterrichts-)Ziele oder Zwecke (Feedback-Umfang und -Funktion), (d) nach welchen Ereignissen im Lernprozess (Feedback-Zeitpunkt) und (e) in welcher Form und mit welchen Darstellungsweisen (Feedback-Darstellung). — (Narciss 2012)

Demnach beschreibt eine Feedback-Strategie die didaktisch begründete Kombination und Gestaltung der Feedbackdimensionen in Abhängigkeit von

- **Situativen und individuellen Faktoren** (z. B. Vorwissen, Aufgabenschwierigkeit)
- **Art und Umfang des Feedbackinhalts**
- **Angestrebter Funktion** (z. B. kognitive Unterstützung, metakognitive Anregung)
- **Zeitpunkt der Bereitstellung**
- **Präsentationsform**

welche in Abbildung 2.1 zusammengefasst sind (Narciss 2006).

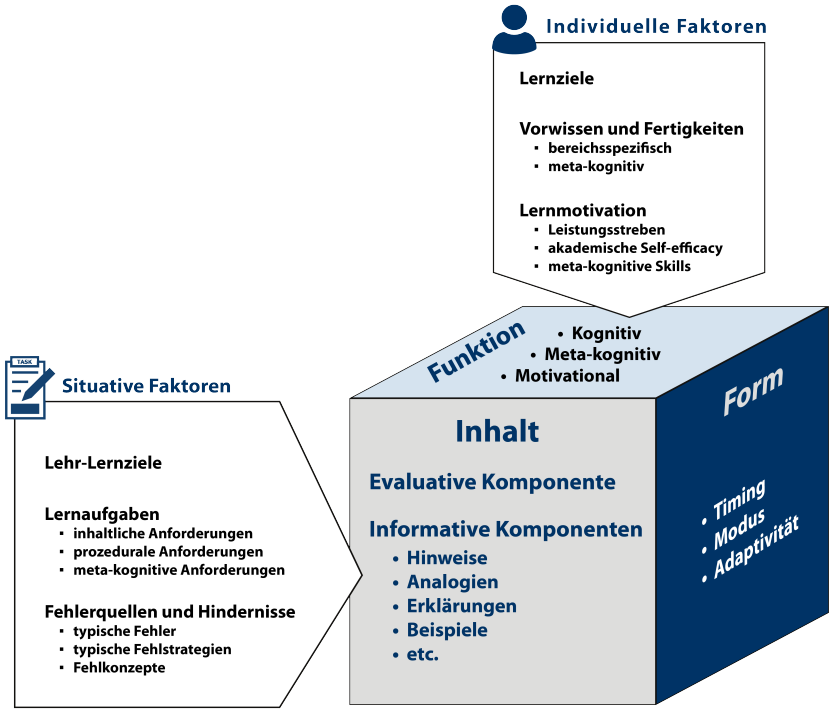


Abbildung 2.1: Relevante Faktoren bei Gestaltung und Evaluation von Feedback-Strategien (Narciss 2006)

Diese fünf Komponenten bilden die Grundlage für die Konzeption differenzierter Feedback-Strategien, die systematisch auf die jeweiligen Bedingungen abgestimmt werden können. Damit wird postuliert, Feedbackdesign nicht nur aus der Perspektive der Botschaft zu denken, sondern auch aus der Perspektive des Kontexts und der Empfängerinnen und Empfänger.

Moderne digitale Lernumgebungen eröffnen hierbei neue Möglichkeiten, Feedback-Strategien gezielt und adaptiv umzusetzen. Allerdings zeigt sich in der Praxis, dass viele Systeme bislang

vorrangig einfache, evaluative Rückmeldungen bereitstellen (z. B. korrekt/inkorrekt-Angaben oder Musterlösungen), während komplexe, auf Forschungserkenntnissen basierende Feedback-Strategien bislang nur begrenzt umgesetzt werden (Keuning, Jeuring und Heeren 2019; Narciss 2020).

Die theoretische Fundierung und systematische Gestaltung effektiver Feedback-Strategien ist daher ein zentrales Anliegen der didaktischen Feedbackforschung. Ein umfassendes Rahmenmodell zur Konzeption und Analyse solcher Strategien stellt das *Interactive Two Feedback Loop (ITFL)*-Modell von Narciss (2020) dar. Dieses Modell integriert die zuvor diskutierten Dimensionen und ordnet sie in ein instruktionspsychologisches Gesamtkonzept ein.

2.1.5 Das Interactive Two Feedback-Loops Model

Das ITFL-Modell (kurz für Interactive Two Feedback-Loops-Model) bietet eine theoretisch fundierte Grundlage für die Gestaltung und Evaluation informativer tutorieller Feedback-Strategien und integriert Erkenntnisse aus der Instruktionspsychologie, systemtheoretische Überlegungen sowie zentrale Befunde der Feedbackforschung (Narciss 2007; Narciss 2020).

Das Modell ist im experimentell- und instruktionspsychologischen Kontext verortet und greift dabei auf eine Vielzahl etablierter Feedbackmodelle zurück. Konkret baut es konzeptionell auf den Arbeiten von Kulhavy und Stock (1989), Bangert-Drowns et al. (1991), Butler und Winne (1995) sowie auf der Metaanalyse von Kluger und DeNisi (1996) auf.

Dabei erweitert es bestehende Ansätze um eine systematische Perspektive, die sowohl kognitive, motivationale als auch metakognitive Aspekte des Lernens berücksichtigt. Ziel ist dabei, Feedback nicht als isolierte Rückmeldung, sondern als interaktiven, regulierenden Bestandteil von Lehr-Lernprozessen zu modellieren.

2.1.5.1 Das Prinzip des Regelkreises

Das ITFL-Modell konzeptualisiert Feedback als Bestandteil eines *dynamischen Regelkreises*, der Lernprozesse kontinuierlich steuert und reguliert (Narciss 2020). Die Grundidee basiert auf einem systemtheoretischen Verständnis von Lernen: Lernprozesse werden als regulierbare Systeme betrachtet, in denen Zielgrößen (SOLL-Werte) mit aktuellen Leistungen (IST-Werten) abgeglichen und durch geeignetes Feedback angepasst werden.

Bevor das ITFL-Modell im Detail dargestellt wird, soll zunächst das diesem Ansatz zugrunde liegende Konzept des *Regelkreises* erläutert werden. Abbildung 2.2 veranschaulicht den schematischen Aufbau eines solchen Kreislaufs, der die zentralen Komponenten und deren Wechselwirkungen zeigt. In der Systemtheorie bezeichnet ein Regelkreis ein geschlossenes Steuerungssystem, das darauf ausgelegt ist, eine vorgegebene Führungsgröße (SOLL-Wert) durch kontinuierliche Rückkopplung zu erreichen und aufrechtzuerhalten (Lunze 2010; Ditton 2014). Die zu beeinflussende Regelgröße stellt dabei den aktuellen Systemzustand dar, der durch geeignete Maßnahmen möglichst nahe an den SOLL-Wert angepasst werden soll. Um dies zu ermöglichen, wird der aktuelle Zustand – also der IST-Wert der Regelgröße – zunächst durch ein *Messglied* erfasst. Dieses Messglied übergibt den ermittelten Wert an die sogenannte *Regeleinrichtung* weiter, in welcher der zentrale SOLL-IST-Vergleich erfolgt. Hier wird durch das *Vergleichsglied* der aktuelle IST-Wert mit dem angestrebten SOLL-Wert verglichen und eine mögliche Abweichung erfasst. Auf dieser Grundlage bestimmt ein *Regler*, welche Maßnahme notwendig ist, um die jeweilige Abweichung zu verringern oder zu beheben. Die geplante Maßnahme wird in Form einer sogenannten *Stellgröße* weitergegeben. Die Stellgröße dient hierbei als Information oder Signal, das durch das Stellglied umgesetzt wird – also durch jene Komponente, die aktiv auf das System einwirkt und es verändert. Die Wirkung entfaltet sich innerhalb der sogenannten *Regelstrecke*, also jenem Bereich,

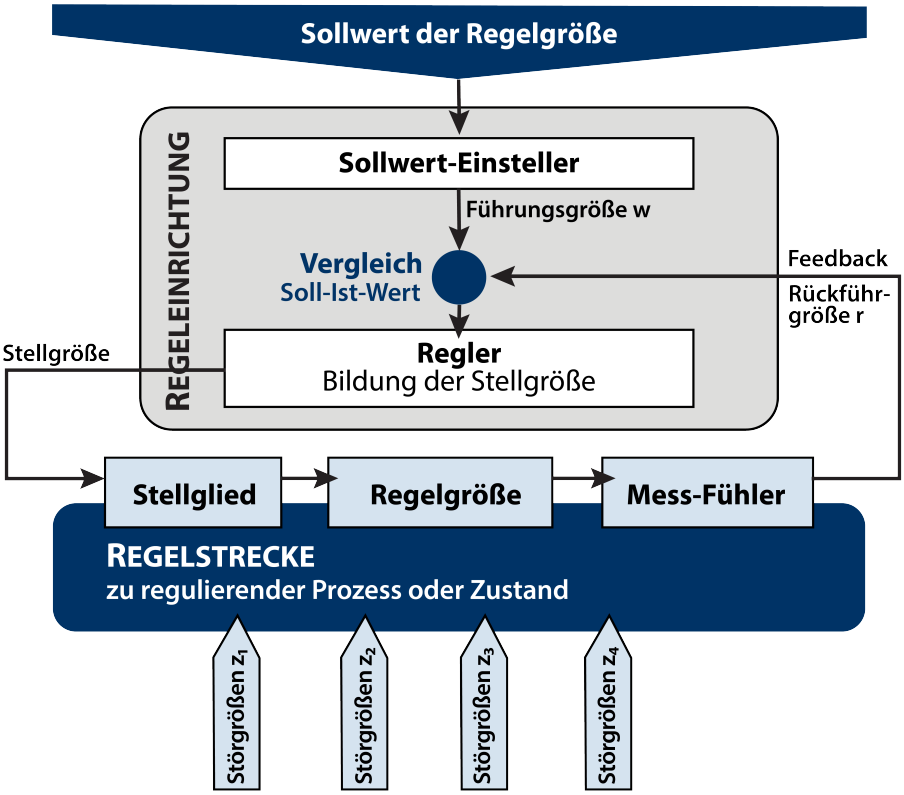


Abbildung 2.2: Schema eines Regelkreises (Narciss 2006)

in dem durch die Eingriffe Veränderungen an der Regelgröße herbeigeführt werden. Sobald diese Veränderungen eintreten, beginnt der Zyklus erneut: Der momentane IST-Wert wird gemessen, verglichen und gegebenenfalls korrigiert.

2.1.5.2 Feedback als regulatives System

Vor diesem Hintergrund lässt sich das ITFL-Modell als spezialisierte Anwendung dieses Regelkreis-Konzepts im Kontext von Lehr-Lern-Prozessen verstehen. Der Lernprozess selbst fungiert dabei als Regelstrecke, in der durch kognitive, metakognitive und motivationale Aktivitäten Veränderungen angestoßen werden. Die Führungsgröße entspricht einem angestrebten Lernziel oder Kompetenzniveau, das es zu erreichen gilt. Die Regelgröße ist in diesem Fall der aktuelle Lern- oder Kompetenzstand der lernenden Person, der durch Aufgabenbearbeitung sichtbar wird. Im ITFL-Modell werden dabei zwei interagierende Regelkreise unterschieden (siehe Abbildung 2.3):

Ein **interner Regelkreis** (auch: **interner Feedback-Loop**), der die selbstgesteuerte Regulation durch die lernende Person modelliert (Selbstbeobachtung, Selbstbewertung, Reflexion, Planung, Strategieanpassung), und ein **externer Regelkreis** (auch: **externer Feedback-Loop**), der die von außen bereitgestellten Feedbackprozesse umfasst (z. B. durch Lehrpersonen, Peers oder Systeme).

Beide Regelkreise operieren dabei auf denselben grundlegenden Prinzipien – dem Vergleich von IST- und SOLL-Zuständen – unterscheiden sich jedoch hinsichtlich der Perspektive, der Messung und der Art der Einflussnahme. Das Modell geht davon aus, dass effektives Feedback immer in einem Zusammenspiel beider Regelkreise wirkt. Es erklärt, wie kognitive, motivationale und metakognitive Prozesse durch passende Rückmeldungen angeregt und reguliert werden können.

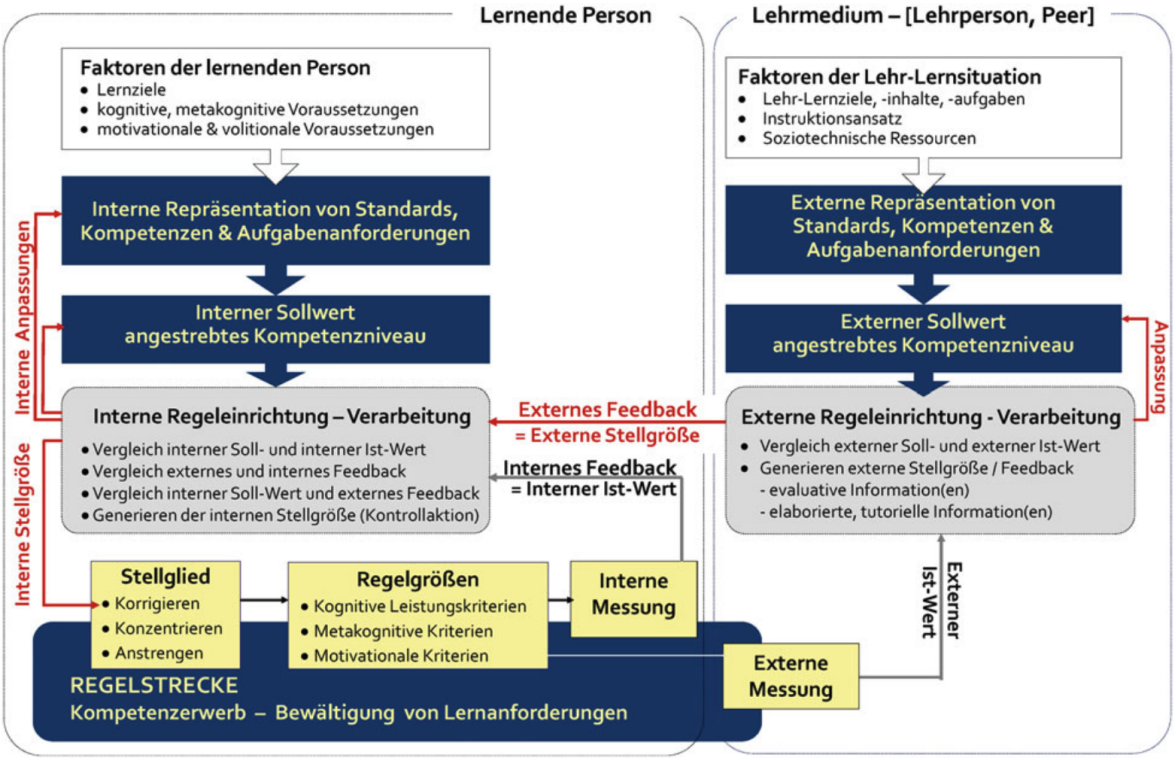


Abbildung 2.3: Zwei interagierende Regelkreise im ITFL-Modell (Narciss 2020)

Im **internen Feedback-Loop** generieren Lernende ein internes Feedback auf Basis subjektiver Zielvorstellungen. Sie verarbeiten Aufgaben, bewerten ihre Lösung und aktivieren ggf. korrigierende Strategien. Hier spielen metakognitive und motivationale Faktoren eine wichtige Rolle. Diskrepanzen zwischen tatsächlicher und angestrebter Leistung können gezielt reflektiert und bearbeitet werden.

Parallel dazu verarbeitet eine externe Feedbackquelle (z. B. Lehrkraft, Lernplattform, LLM-basiertes System) im **externen Feedback-Loop** eine beobachtbare Leistung, identifiziert Diskrepanzen zum erwarteten SOLL-Zustand und stellt entsprechendes (externes) Feedback bereit welches dann vom Lernenden in seinen internen Regelprozess integriert werden kann.

Wichtig: Der Effekt des externen Feedbacks hängt maßgeblich davon ab, ob und wie es in den internen Regelkreis der Lernenden integriert wird – was wiederum stark von individuellen Faktoren wie Vorwissen, Zielorientierung und Feedbackkompetenz abhängt.

Das ITFL-Modell verdeutlicht, dass Feedback weit mehr als nur eine Mitteilung über Korrektheit ist. Es ist eine dynamische Interaktion zwischen Lernenden und ihrer Umwelt, die über mehrere kognitive und metakognitive Ebenen hinweg regulierend wirken kann. Die systematische Integration von Funktion, Inhalt, Form, sowie individueller und situativer Bedingungen eröffnet dabei eine Vielzahl an Möglichkeiten zur Gestaltung von Feedback, insbesondere in digitalen Lernumgebungen. Möchte man dies automatisiert unterstützen, so ist es notwendig, die unterschiedlichen Dimensionen von Feedback mitsamt ihren Ausprägungsformen zu modellieren und in ein kohärentes System zu überführen. Diese Modellierung bildet die Grundlage für die Entwicklung von Lernsystemen, die nicht nur einfaches Feedback geben, sondern komplexe, kontextsensitive Rückmeldungen bereitstellen können, welche auf die individuellen Lernvoraussetzungen und Bedürfnisse der Lernenden abgestimmt sind.

2.2 Aufgaben als Grundlage für Feedbackprozesse

Die vorangehenden Ausführungen haben verdeutlicht, dass Feedbackprozesse wesentlich darin bestehen, Diskrepanzen zwischen einem aktuellen IST- und einem angestrebten SOLL-Zustand zu verringern, indem durch geeignete Rückmeldungen regulierend in den Lernprozess eingegriffen wird (Narciss 2020). Damit ein solcher Vergleich überhaupt möglich ist, bedarf es einer Referenz, an der Lernhandlungen sichtbar, bewertbar und interpretierbar werden. Diese Referenz bilden *Aufgaben*: Sie strukturieren Lernaktivitäten, erzeugen Bearbeitungsspuren und eröffnen damit die diagnostische Grundlage, auf der Feedbackprozesse aufbauen können. Aufgaben sind somit das zentrale Bindeglied zwischen Lernaktivität und Feedbackprozess.

Vor diesem Hintergrund widmet sich das folgende Kapitel der theoretischen Fundierung von (Lern-)aufgaben. Es wird herausgearbeitet, welche grundlegenden Funktionen Aufgaben im Lernprozess übernehmen und welche Anforderungen sich daraus für den Einsatz in adaptiven Feedbacksystemen ergeben.

2.2.1 Funktionen und Wirksamkeit von Lernaufgaben

In vielen didaktischen Konzepten gelten Aufgaben als zentrales Mittel, um Lernprozesse anzuregen, zu strukturieren und zu überprüfen (Kleinknecht et al. 2013; Ruf, Berges und Hubwieser 2015). Sie ermöglichen es Lernenden, sich aktiv mit fachlichen Inhalten auseinanderzusetzen – etwa durch Anwendung, Transfer, Problemlösung oder Reflexion – und leisten damit einen entscheidenden Beitrag zum individuellen Kompetenzaufbau. In lernpsychologischer Perspektive wird Lernen nicht als bloßer Informationsaufnahmeprozess verstanden, sondern als aktiver Konstruktionsprozess, bei dem Wissen durch zielgerichtete Auseinandersetzung mit Anforderungen aufgebaut, vernetzt und modifiziert wird (Piaget 2003; Bruner 1966; Vygotsky 1980). Lernaufgaben übernehmen in diesem Prozess eine zentrale Rolle, da sie nicht nur Wissen abfragen, sondern Lernprozesse initiieren, anleiten und sichtbar machen können (Merrill 2002; Anderson und Krathwohl 2001; Bransford, Brown und Cocking 2000).

Körndle, Narciss und Proske (2004) betonen, dass Aufgaben besonders dann lernwirksam sind, wenn sie systematisch konstruiert und auf psychologischen Prinzipien begründet sind. Sie können verschiedene Funktionen im Lernprozess erfüllen: von der Aktivierung von Vorwissen über den Erwerb und die Anwendung neuen Wissens bis hin zur gezielten Übung und Diagnose. Im kompetenzorientierten Unterricht stellt die Auswahl und Entwicklung geeigneter Aufgaben daher eine zentrale Herausforderung dar. In der fachdidaktischen Forschung gilt die gezielte Gestaltung solcher Aufgaben als wesentlicher Bestandteil des pädagogischen Wissens von Lehrpersonen (Hubwieser et al. 2013).

Auf dieser Grundlage lässt sich die doppelte Rolle von Aufgaben verdeutlichen: Einerseits regen sie kognitive und metakognitive Prozesse an und leisten damit einen wesentlichen Beitrag zum Kompetenzaufbau. Andererseits erzeugen sie diagnostische Daten, die eine Grundlage für Feedbackentscheidungen bilden. Diese Doppelrolle macht sie für adaptive Systeme besonders relevant, da sie sowohl Lerngelegenheiten bereitstellen als auch die für eine Soll-Ist-Analyse notwendigen Bearbeitungsspuren sichtbar machen.

2.2.2 Definition von Aufgaben

Die bislang dargestellten Funktionen von Aufgaben im Lernprozess lassen erkennen, dass sie stets eine Transformation anstoßen: Lernende sollen von einem gegebenen Ausgangszustand zu einem intendierten Zielzustand gelangen. Um diese Rolle präziser zu fassen, lohnt ein Blick auf die begriffliche Bestimmung von Aufgaben im didaktischen und testtheoretischen Diskurs.

Im allgemeinen Verständnis werden Aufgaben als Kombination aus einer Handlungsaufforderung und der erwarteten Bearbeitung durch den Lernenden beschrieben (Schabram 2007). In klassischen Modellen der Test- und Aufgabenkonstruktion erscheinen sie als Verknüpfung eines gegebenen Problems (*Stimulus*) mit einer erwarteten Handlung oder Antwort (*Response*) (Klauer 1987).

Für den Kontext der Programmierausbildung konkretisieren Ruf, Berges und Hubwieser (2015) diesen funktionalen Aufgabenbegriff. Sie verstehen Aufgaben als Anweisung (*to do*), einen gegebenen (Ausgangs-)zustand (*given problem*) in einen gewünschten Zielzustand (*expected solution*) zu transformieren. Wissen ist dabei sowohl im Ausgangszustand als auch im Zielzustand repräsentiert und wird durch den Einsatz kognitiver Prozesse in den Zielzustand überführt (Ruf, Berges und Hubwieser 2015). Diese Perspektive verbindet die inhaltliche Struktur mit den erforderlichen Denkprozessen und verdeutlicht, wie Aufgaben sowohl Lernhandlungen initiieren als auch diagnostische Anknüpfungspunkte für Feedback bereitstellen.

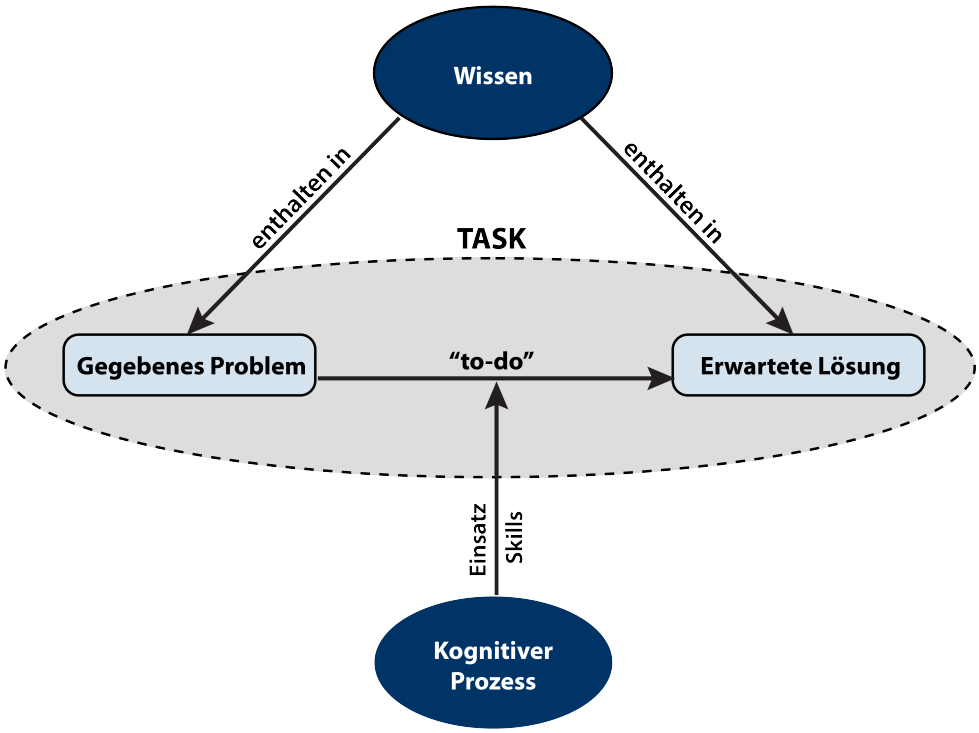


Abbildung 2.4: Rollen von Wissen und kogn. Prozessen beim Lösen einer Aufgabe (Ruf, Berges und Hubwieser 2015)

Aufgaben formulieren somit nicht nur Anforderungen, sondern stellen auch einen Referenzrahmen bereit, an dem Bearbeitungen bewertet und Feedbackprozesse ausgerichtet werden können. Damit bildet dieser Aufgabenbegriff die konzeptuelle Grundlage für die in Kapitel 5 vorgestellte Formalisierung von Programmieraufgaben, die wiederum die technische Basis für die automatisierte Generierung von kontextsensitivem Feedback darstellt.

2.2.3 Klassifizierung von Aufgaben

Ein etabliertes Kriterium zur Klassifikation ist die *Funktion* der Aufgabe im Lernprozess. Kleinknecht et al. (2013) unterscheiden dabei vier grundlegende Typen:

1. **Lernaufgaben**, die auf Aufbau neuen Wissens und grundlegender Kompetenzen abzielen,
2. **Übungsaufgaben**, zur Festigung und Automatisierung bereits eingeführter Inhalte,
3. **Anwendungsaufgaben**, die den Transfer und die Vernetzung von Wissen fördern,
4. **Diagnostische Aufgaben**, die der Sichtbarmachung des aktuellen Lernstands dienen.

Diese Aufgabenformen lassen sich gemeinsam als **lernorientierte Aufgaben** fassen, da sie primär der Förderung, Strukturierung und Diagnose von Lernprozessen dienen. Demgegenüber stehen **Leistungsaufgaben**, die nicht unmittelbar auf Lernförderung ausgerichtet sind, sondern vor allem der summativen Bewertung. Sie werden typischerweise unter standardisierten Bedingungen und ohne Hilfsmittel eingesetzt, um den erreichten Wissens- oder Kompetenzstand der Lernenden zu bewerten.

Die Abgrenzung ergibt sich somit weniger aus dem Zeitpunkt der Aufgabenstellung (vor, während oder nach dem Lernen), sondern vor allem aus deren Gestaltung und intendierter Funktion. Während lernorientierte Aufgaben vielfältige Unterstützungsangebote enthalten und zur Reflexion anregen können (Körndle, Narciss und Proske 2004), fokussieren Leistungsaufgaben auf eine Leistungsmessung.

Über die reine Funktion hinaus lässt sich das *kognitive Aktivierungspotenzial* von Aufgaben differenziert beschreiben. Das Klassifikationsmodell von Kleinknecht et al. (2013) schlägt hierzu mehrere Kriterien vor:

- die Art des angesprochenen Wissens (faktisch, prozedural, konzeptuell),
- die Anzahl und Komplexität der Wissenseinheiten,
- die Offenheit der Aufgabe,
- den Lebensweltbezug,
- die sprachlogische Komplexität sowie
- die verwendeten Repräsentationsformen.

Zentral ist dabei die Kombination aus kognitiver Anforderung, Wissensdimension und Kontextmerkmalen. Die erste Kategorie differenziert Aufgaben nach dem angesprochenen Wissenstyp und orientiert sich an den Wissensdimensionen der revidierten Bloom'schen Taxonomie (Anderson und Krathwohl 2001). Der geforderte kognitive Prozess lässt sich in allgemeinen Kategorien wie Reproduktion, enger Transfer, weiter Transfer und Problemlösung verorten. Darüber hinaus bestimmen die Anzahl und Komplexität der eingebundenen Wissenselemente die wahrgenommene Schwierigkeit einer Aufgabe, ebenso wie der Fokus oder die Detailliertheit der Aufgabenbeschreibung. Motivationale Aspekte werden schließlich durch den Lebensweltbezug sowie durch die Vielfalt der verwendeten Darstellungsformen beeinflusst (Kleinknecht et al. 2013).

Für Programmieraufgaben wurde ein spezialisiertes Klassifikationsschema von Ruf, Berges und Hubwieser (2015) entwickelt. Es unterscheidet Aufgaben auf Basis (1) der **Aufgabenstellung** (z. B. Lückencode, Debugging, Freitext), (2) der **erwarteten Lösungsform** (z. B. Codegenerierung, Codeverständnis), und (3) des **Übergangsprozesses** von Problemstellung zur Lösung. Sie identifizieren elf typische Programmieraufgabentypen, deren Offenheit und Komplexität stark variieren und postulieren: Je weniger Vorgaben eine Aufgabe macht, desto größer ist der Interpretationsspielraum und desto höher die kognitive Aktivierung.

2.3 Adaptive Lernsysteme (ALS)

Bestrebungen, Feedback zu automatisieren, existieren seit den 1920er Jahren (Benjamin 1988). Die Entwicklung solcher Systeme wurde nicht zuletzt durch die zunehmende Verbreitung computerbasierter Lernsysteme sowie die wachsende Nachfrage nach skalierbaren Bildungsangeboten motiviert. Ziel solcher Systeme war und ist es, Lernende durch automatisierte Rückmeldungen zu unterstützen und dabei individuelle Lernverläufe adaptiv zu begleiten (Sleeman und Brown 1982; Koedinger et al. 1997).

Die Ausführungen in Abschnitt 2.1 haben gezeigt: Die Herausforderung, qualitativ hochwertiges Feedback automatisiert bereitzustellen, erweist sich als hochkomplex. Feedback entfaltet seine Wirksamkeit nicht allein durch die Korrektur von Fehlern, sondern durch eine differenzierte, kontextualisierte und didaktisch angemessene Unterstützung individueller Lernprozesse. Hinzu kommt die hohe Heterogenität von Lernendengruppen: Unterschiede im Vorwissen, in Denkstrategien sowie in Lernpräferenzen führen dazu, dass identische Aufgabenstellungen sehr unterschiedliche Reaktionen und Fehlerbilder hervorrufen können. Besonders bei offenen Aufgabenformaten wie sie u. a. in der Programmierausbildung vorkommen, ist die Variabilität möglicher Lernendenantworten erheblich.

Während erfahrene Lehrpersonen in der Lage sind, flexibel und situativ auf individuelle Lernstände und Antworten zu reagieren, stützen sich Feedback-Entscheidungen häufig auf implizites Erfahrungswissen. Wann welches Feedback in welcher Form gegeben werden sollte, ist weder Teil standardisierter Aus- und Weiterbildungsprogramme für Lehrkräfte noch in der Forschung abschließend geklärt (Ditton 2014). Feedback-Strategien werden meist erst im Verlauf der Lehr-Praxis durch wiederkehrende Handlungsroutinen und implizite Fallanalysen entwickelt. Die Herausforderung, solche dynamischen und oft intuitiven Entscheidungen algorithmisch zu rekonstruieren, macht deutlich, dass die Automatisierung von Feedbackprozessen mehr erfordert als nur technische Lösungen. Es bedarf vielmehr einer systematischen Analyse, Modellierung und Formalisierung jener Prozesse, die erfahrene Lehrpersonen implizit ausführen. Erst auf dieser Grundlage lassen sich adaptive Systeme entwickeln, die in der Lage sind, Lernprozesse individuell zu begleiten und lernförderliche Rückmeldungen bereitzustellen.

2.3.1 Definition und Einordnung

Bevor zentrale Entwicklungen, Paradigmen und Technologien adaptiver Lernsysteme im Verlauf dieses Kapitels näher beleuchtet werden, soll zunächst eine begriffliche Klärung vorgenommen werden. Insbesondere bedarf es einer systematischen Unterscheidung zwischen allgemeinen Lernumgebungen, computergestützten Lernsystemen und dem spezifischen Konzept der Adaptivität.

Unter einer *Lernumgebung* wird allgemein jede didaktisch gestaltete Konstellation verstanden, in der Lernprozesse angestoßen und unterstützt werden können. Dazu gehören Aufgabenstellungen, Materialien, Rückmeldeformate sowie soziale und technische Rahmenbedingungen (Wilson 1996). Lernumgebungen können analog oder digital realisiert sein und reichen von streng instruktionsorientierten Formaten bis hin zu offenen, explorativen Settings.

Mit der Digitalisierung des Bildungsbereichs entstanden zunehmend *computerbasierte Lernumgebungen*, die nicht nur der Inhaltspräsentation, sondern auch der Interaktionsgestaltung und Rückmeldung dienen. Erste Systeme orientierten sich dabei häufig am behavioristischen Paradigma, etwa in Form von programmiertem Unterricht, bei dem der Lernfortschritt linear durch strukturierte Aufgabenfolgen gesteuert wurde (Ditton 2014). Spätere Entwicklungen griffen stärker kognitivistische Ansätze auf, indem sie Lernende als aktive Informationsverarbeiter verstanden und adaptive Unterstützungsformen in den Blick nahmen (siehe hierzu Abschnitt 2.3.2).

Digitale *Lernsysteme* im engeren Sinne gehen über die bloße Bereitstellung von Inhalten hinaus: Sie erfassen Lernaktivitäten, analysieren Bearbeitungsverläufe und bieten automatisierte Rückmeldungen. Typische Komponenten sind Module zur Aufgabenpräsentation, Eingabeanalyse, Feedbackgenerierung sowie gegebenenfalls zur Leistungserfassung oder Lernstandsdokumentation. Lernsysteme bilden damit das technische Fundament für die Operationalisierung didaktischer Prinzipien im digitalen Raum.
Allerdings sind nicht alle Lernsysteme per se *adaptiv*. In vielen Fällen erfolgen Rückmeldungen statisch – etwa durch die Anzeige eines Lösungshinweises oder eines Punktestands, unabhängig vom individuellen Lernprozess (Keuning, Jeuring und Heeren 2016). Auch automatische Bewertungssysteme (*grading systems*) zählen hierzu: Sie dienen primär der Beurteilung von Lernleistungen und basieren häufig auf vordefinierten Musterlösungen oder formalen Kriterien, ohne dass eine tiefere Prozessdiagnose oder Anpassung erfolgt (Aldriye, Alkhalaf und Alkhalaf 2019).

Was macht ein Lernsystem nun aber *adaptiv*? Digitale Lernsysteme gelten dann als *adaptiv*, wenn sie ihre Funktionsweise dynamisch an Merkmale, Reaktionen oder Lernverläufe der Lernenden anpassen können. Im Kontext der Programmierung lassen sich in Lernsystemen dabei zwei grundlegende Formen der Adaptivität unterscheiden – **adaptive Navigation** und **adaptives Feedback** (Crow, Luxton-Reilly und Wuensche 2018; Le 2016):

Adaptive Navigation bezieht sich auf die strukturierende Funktion eines Systems: Lernpfade, Aufgabenreihenfolgen oder Schwierigkeitsgrade werden basierend auf dem diagnostizierten Lernstand individuell angepasst. Ziel ist es hier z. B., Lernenden Inhalte in einer Form und Reihenfolge zu präsentieren, die optimal an ihren jeweiligen Wissensstand anschließt. So verwendet das System ELM-ART beispielsweise Pfadsteuerungen auf Grundlage kognitiver Modelle und Erfolgsindikatoren aus bisherigen Aufgabenbearbeitungen (Crow, Luxton-Reilly und Wuensche 2018).

Adaptives Feedback hingegen bezieht sich auf Art, Umfang und Zeitpunkt von Rückmeldungen innerhalb des Lernprozesses. Dabei liegt der Fokus häufig nicht auf evaluativem Feedback, sondern auf der Bereitstellung von elaborierenden Feedback-Typen – abhängig vom individuellen Antwortverhalten und dem jeweiligen Lernstand (Le 2016).

Adaptivität setzt somit voraus, dass das System Informationen über den aktuellen Zustand des Lernenden diagnostisch verarbeitet – sei es durch regelbasierte Entscheidungslogik oder durch datengetriebene Verfahren wie maschinellem Lernen. Entscheidend ist, dass die Rückmeldung nicht nur bewertend, sondern steuernd in den Lernprozess eingreift. In diesem Sinne schließt Adaptivität sowohl formative Funktionen wie etwa adaptive Hinweise als auch strukturelle Anpassungen wie die dynamische Gestaltung von Aufgabenpfaden ein (Aleven et al. 2017).

Während adaptive Lernsysteme darauf ausgelegt sind, Lernprozesse kontinuierlich zu begleiten und personalisierte Unterstützungsmaßnahmen bereitzustellen, existieren eine Vielzahl digitaler Systeme, die primär auf die Beurteilung von Leistungen fokussieren (Strickroth und Striewe 2022). Dazu zählen etwa Systeme, die eingereichten Programmcode anhand vordefinierter Testfälle analysieren und bewerten. Vor diesem Hintergrund ist eine begriffliche und funktionale Abgrenzung zu sogenannten *Grading-Systemen* erforderlich, deren Hauptzweck in einer reinen Leistungsbewertung liegt.

Auch wenn Grading-Systeme Rückmeldungen bereitstellen, erfüllen sie nicht die Kriterien eines adaptiven Lernsystems: Sie reagieren in der Regel nicht dynamisch auf individuelle Lernverläufe, initiieren meist keine gezielten Interventionen zur Behebung von Fehlern und bieten keine prozessbegleitende Unterstützung. Stattdessen liefern sie standardisierte, meist nicht personalisierte Rückmeldungen, bei denen die Bewertung und nicht die Lernförderung im Fokus steht.

Vor diesem Hintergrund erscheint es auch begrifflich fragwürdig, hier von *Feedback* im engeren Sinne zu sprechen. Da keine Absicht besteht, eine bestehende Differenz zwischen aktuellem Zu-

stand und angestrebtem Ziel systematisch zu adressieren, sondern lediglich der IST-Zustand rückgemeldet wird, ist die Bezeichnung *summative Rückmeldung* treffender (siehe dazu Abschnitt 2.1.1). Sie verweist auf eine abschließende Mitteilung über das erreichte Leistungsniveau im Verhältnis zu einem erwarteten Ergebnis – ohne gezielte Anregung zur Verbesserung oder Weiterarbeit. Gleichwohl ist anzumerken, dass einige moderne Grading-Systeme über rein summative oder evaluative Rückmeldungen hinausgehen. So finden sich zunehmend rudimentäre Feedbackelemente – etwa Hinweise auf typische Fehler oder exemplarische Lösungsvorschläge. Für die Einordnung im Rahmen dieser Arbeit ist jedoch das grundlegende Ziel des Systems ausschlaggebend: Liegt der primäre Fokus auf der Leistungsbewertung und nicht auf der prozessbegleitenden Unterstützung von Lernenden, so handelt es sich nicht um ein adaptives Lernsystem – auch wenn vereinzelte feedbackähnliche Komponenten integriert sind. Die zentrale Eigenschaft adaptiver Systeme bleibt die kontinuierliche, differenzierende Reaktion auf individuelle Lernverläufe auf Basis fortlaufender Diagnose.

In Anlehnung an Paramythis und Loidl-Reisinger (2004) soll im Rahmen dieser Dissertation die folgende Arbeitsdefinition für *adaptive Lernsysteme* (ALS) gelten, welche sowohl klassische regelbasierte Systeme als auch moderne datenbasierte Ansätze einschließt:

Adaptives Lernsystem (ALS)

Ein digitales Lernsystem, das diagnostisch gewonnene Informationen über Lernende oder deren Antworten nutzt, um Rückmeldungen, Aufgabenstellungen oder Lernpfade gezielt und dynamisch an individuelle Lernprozesse anzupassen.

2.3.2 Lernsysteme im Wandel der Zeit

Bereits im Jahr 1924 zeigte Sidney Pressey mit seiner mechanischen „Lehrmaschine", dass automatisierte Rückmeldung grundsätzlich möglich ist: Das Gerät präsentierte den Lernenden Multiple-Choice-Fragen über eine mechanische Anzeigevorrichtung. Eine richtige Antwort ermöglichte das automatische Fortschalten zur nächsten Frage, während falsche Eingaben blockiert wurden. Zusätzlich wurde ein Punktestand angezeigt, der über die Anzahl korrekt beantworteter Fragen informierte (Petrina 2004). Diese Form von Rückmeldung würde heute als `KP`-Feedback eingestuft werden, da sie über den allgemeinen Leistungsstand informiert, ohne weiterführende inhaltliche Hinweise zu geben. In ihrer Einfachheit demonstrierte Presseys Maschine jedoch bereits das Grundprinzip adaptiver Rückmeldung: eine unmittelbare Reaktion auf Lernendenverhalten mit dem Ziel, Lernprozesse zu strukturieren und zu steuern.

Eines der ersten computerbasierten Lernsysteme war PLATO (Programmed Logic for Automated Teaching Operations), das ab 1960 an der University of Illinois entwickelt wurde. Mit dem technischen Fortschritt und der zunehmenden Verfügbarkeit elektronischer Rechensysteme eröffnete PLATO neue Möglichkeiten zur Gestaltung und Skalierung von Bildungsprozessen. Über Terminalzugänge konnten Lernende auf Unterrichtseinheiten zugreifen, interaktive Übungen bearbeiten und sofort Rückmeldung erhalten (Smith und Sherwood 1976; Cope und Kalantzis 2023). Besonders war hier die Bandbreite an Funktionalitäten, die über reine Inhaltsvermittlung hinausging: PLATO ermöglichte beispielsweise Online-Tests mit unmittelbarem Korrekturfeedback, Punktestände zur Leistungsrückmeldung (`KP`-Feedback), sowie erklärende Hinweise bei Fehlern (`KM`). Auch Austauschformate wie Foren, Nachrichten oder Chat wurden früh integriert — Funktionen, die heute als Feedbackquellen im Sinne sozialer Interaktion (z. B. Peer- oder Lehrkräftefeedback) verstanden werden können. PLATO legte damit die Grundlage für erste computervermittelte Feedbackprozesse, insbesondere im Bereich von *Knowledge of Result (KR)*, *Knowledge of Performance (KP)* und erste Ansätze von *Knowledge about Mistakes (KM)*. Obwohl noch keine echte

Adaptivität, wie man sie in modernen Plattformen definieren würde, vorlag, wurde mit PLATO der Versuch unternommen, Feedbackstrukturen technisch abzubilden, die zuvor ausschließlich Lehrpersonen vorbehalten waren.

Mit der Verbreitung erschwinglicher Homecomputer setzte in den 1980er-Jahren ein nachhaltiger Entwicklungsschub im Bereich computergestützter Lernsysteme ein. Aufbauend auf früheren Ansätzen entstand eine neue Generation von Systemen, die nicht nur Inhalte präsentieren, sondern aktiv den Lernprozess begleiten und auf individuelle Lernstände reagieren sollten. Diese Systeme wurden unter dem Begriff *Intelligente Tutoring-Systeme* (ITS) bekannt, deren zentrales Ziel es ist, das Verhalten menschlicher Tutorinnen und Tutoren zu simulieren. Technisch wurde dies auf Basis kognitionspsychologischer Modelle und wissensbasierter Ansätze umgesetzt – etwa durch Inferenzregeln, welche Domänenwissen formal repräsentieren. Eine ausführlichere Darstellung der grundlegenden Architekturen sowie Komponenten erfolgt in Abschnitt 2.3.3.1. Ein prominentes Beispiel aus dieser Phase ist der *LISP Tutor*, der speziell auf die Unterstützung beim Erlernen von Programmiersprachen ausgerichtet war und gezielt kontextuelle Hilfestellungen generierte (Farrell, Anderson und Reiser 1984; Anderson et al. 1995). ITS unterschieden sich damit wesentlich von den vorangegangenen Systemen, die überwiegend lineare Instruktionspfade ohne Rückbezug auf individuelle Lernprozesse boten (Nwana 1990).

In der Anfangszeit dominierten stark regelbasierte Systeme, die durch ihre Transparenz und Nachvollziehbarkeit überzeugten. Diese Systeme stützten sich auf manuell kodiertes Expertenwissen in Form von If-Then-Regeln, waren jedoch nur begrenzt skalierbar – insbesondere bei offenen Aufgabenformaten oder bei hoher Variabilität in den Lernendenantworten. Ab den 1990er-Jahren zeichnete sich ein grundlegender Paradigmenwechsel ab: Mit der zunehmenden Verfügbarkeit leistungsfähiger Rechentechnik, der wachsenden Menge an Nutzungsdaten aus digitalen Lernsystemen sowie Fortschritten in der Forschung zu maschinellem Lernen begann man zunehmend datengetriebene Ansätze und statistische Modelle zu nutzen. Verfahren wie das *Bayesian Knowledge Tracing* ermöglichten es, individuelle Lernverläufe probabilistisch zu modellieren, Vorhersagen über künftiges Lernverhalten zu treffen und darauf basierend Feedback zu adaptieren (Corbett und Anderson 1995). Damit vollzog sich in der Lernsystemforschung ein Übergang vom symbolisch-regelbasierten zum subsymbolisch-datengetriebenen Paradigma – ein Wandel, der auch den Begriff der Adaptivität neu definierte. Dieser Übergang markierte den Beginn adaptiver Lernsysteme im engeren Sinne: Systeme, die sich nicht mehr nur auf fest kodiertes Expertenwissen stützten, sondern auf der Analyse empirischer Lernprozesse aufbauten – mit dem Ziel, sich dynamisch an individuelle Lernverläufe anzupassen.

Spätestens mit der Veröffentlichung von ChatGPT Ende 2022 haben sich neue Impulse für die Gestaltung adaptiver Lernsysteme ergeben: Die Fortschritte im Natural Language Processing sowie die Leistungsfähigkeit großer Sprachmodelle (Large Language Models, LLMs) eröffnen neue Potenziale für die automatisierte Erstellung kontextsensitiver Rückmeldungen, die dynamische Generierung individualisierter Aufgaben sowie die Umsetzung dialogbasierter Lerninteraktionen (Sarsa et al. 2022; Lohr et al. 2025b). Erste Studien deuten auf potenziell positive Effekte hin (Yilmaz und Karaoglan Yilmaz 2023; Boguslawski, Deer und Dawson 2025; Fan et al. 2025). Gleichwohl ist bislang unklar, unter welchen Bedingungen diese generativen Ansätze tatsächlich lernwirksam sind und wie sie in didaktisch fundierte Systeme integriert werden können.

Aussagen zur tatsächlichen Wirkung solcher Systeme sind daher (zunächst) mit Vorsicht zu betrachten: Generative KI kann dazu beitragen, Feedbackprozesse in großskaligen Lernszenarien effizienter zu gestalten und personelle Ressourcen zu entlasten. Ihre konkreten Effekte auf Lernverhalten und -erfolg sind bislang jedoch nur fragmentarisch empirisch untersucht.

2.3.3 Intelligente Tutoring-Systeme

Bereits in Abschnitt 2.3.2 wurde auf die Entstehung sogenannter *Intelligenter Tutoring-Systeme* hingewiesen, die seit den 1980er-Jahren eine zentrale Rolle in der Entwicklung digitaler Lernsysteme einnehmen. Im Unterschied zu früheren instruktionsbasierten Ansätzen verfolgen ITS das Ziel, ausgewählte Funktionen menschlicher Tutorinnen und Tutoren algorithmisch nachzubilden, um individuelle Lernprozesse differenziert und kontinuierlich zu unterstützen.

ITS sind grundlegend so konzipiert, dass sie Lernendendaten in Echtzeit verarbeiten, Bearbeitungsverläufe analysieren und darauf basierend personalisierte Unterstützungsmaßnahmen bereitstellen können – etwa in Form von Feedback, Hinweisen oder adaptiven Aufgabenstellungen. Damit soll eine Effektivität erreicht werden, die der individuellen Betreuung durch Lehrpersonen möglichst nahekommt, zugleich jedoch digital skalierbar bleibt (Psotka, Massey und Mutter 1988; Crow, Luxton-Reilly und Wuensche 2018). Im Mittelpunkt steht dabei die Annahme, dass effektives Lernen durch eine adaptive Begleitung gefördert wird, die sich an den individuellen Voraussetzungen, Fehlern und Strategien der Lernenden orientiert. ITS operationalisieren dieses Prinzip, indem sie fortlaufend diagnostische Informationen nutzen, um gezielte Interventionen auszulösen und so den Lernprozess aktiv zu steuern (Okpo 2016).

Intelligente Tutoring-Systeme sind damit abzugrenzen von Systemen, welche auf eine reine automatisierte Bewertung abzielen (*Automated Grading Systems*) und damit primär der Leistungsbeurteilung dienen, indem sie Artefakte – wie etwa Antworten oder Programmcode – anhand vordefinierter Kriterien bewerten (siehe auch Abschnitt 2.3.1). Der Fokus liegt dabei auf dem Ergebnis, nicht auf dem Prozess: Bei diesen Systemen erfolgt keine kontinuierliche Lernprozessanalyse und keine gezielte Rückmeldung während der Bearbeitung. Auch wenn solche Systeme mitunter ebenfalls Rückmeldungen liefern, handelt es sich dabei meist um statische Hinweise, die nicht auf individuelle Bearbeitungsverläufe reagieren und die nicht das Ziel verfolgen, Lernprozesse zu begleiten und mögliche Lücken zu schließen – damit nicht um Feedback.

Gleichwohl ist die Fähigkeit zur Analyse lernbezogener Artefakte eine zentrale Voraussetzung für ITS. Denn um adaptives Feedback generieren zu können, müssen zunächst Informationen über den aktuellen Lernstand, typische Fehlermuster oder Bearbeitungsstrategien erhoben werden. Diagnostische Verfahren – ob regelbasiert oder datengetrieben – bilden somit die Grundlage für eine Auswahl oder Generierung lernförderlicher Interventionen. In diesem Sinne sind Beurteilungsprozesse nicht das Ziel, sondern ein notwendiger Zwischenschritt innerhalb adaptiver Unterstützungsmechanismen.

2.3.3.1 Architektur und Komponenten

Um das Ziel individueller und kontextsensitiver Lernunterstützung zu erreichen, haben sich über die Jahre verschiedene Architekturmodelle für Intelligente Tutoring-Systeme herausgebildet. Trotz unterschiedlicher Ausprägungen lassen sich einige zentrale Komponenten identifizieren, die in nahezu allen klassischen und modernen ITS-Ansätzen eine grundlegende Rolle spielen (Padayachee 2002):

Im Kern folgt die Mehrzahl der Systeme der sogenannten *Drei-Komponenten-Architektur*, bestehend aus: (1) einem **Domänenmodell**, das das fachliche Wissen der jeweiligen Lernumgebung strukturiert repräsentiert, (2) einem **Lernendenmodell**, in dem z. B. individuelle Wissensstände, Fehlkonzepte, Bearbeitungsstrategien oder kognitive Merkmale der Lernenden abgebildet werden, sowie (3) einem **Tutoringmodell** (auch *Pedagogical Model*), das Regeln und Strategien zur didaktischen Steuerung des Lernprozesses enthält. Auf Basis der im Lernendenmodell gespeicherten Informationen und der Inhalte des Domänenmodells werden daraus systemseitig Maßnahmen wie die Auswahl von Feedback, Aufgaben oder Hilfestellungen abgeleitet.

Diese klassische Struktur wurde später zur sogenannten *Vier-Komponenten-Architektur* erweitert, in der die **Benutzerschnittstelle** (auch *User Interface*) als eigenständige Komponente betrachtet wird. Sie ist verantwortlich für die mediale Darstellung der Lerninhalte sowie die Gestaltung der Interaktion mit dem System, einschließlich adaptiver Dialogführung und Usability-Aspekten (Dede 1986).

Je nach Zielsetzung und technologischem Rahmen werden diese Basismodelle in aktuellen ITS häufig um weitere Komponenten ergänzt – etwa um Kontextmodelle, soziale Interaktionsmodelle oder Modelle zur Erfassung emotionaler Zustände. Unabhängig von der konkreten Modellarchitektur basiert die Funktionsweise moderner ITS grundlegend auf dem Prinzip der Diagnostik: Lernendenverhalten wird kontinuierlich erfasst und analysiert, um Rückschlüsse auf individuelle Wissensstände, Bearbeitungsstrategien oder Lernfortschritte zu ziehen.

2.3.3.2 Paradigmen der Wissens- und Prozessmodellierung in ITS

Über die architektonischen Grundstrukturen hinaus wurden in der Forschung zu ITS verschiedene Paradigmen entwickelt, die beschreiben, wie fachliches Wissen und Lernprozesse systematisch modelliert werden können. Nkambou, Bourdeau und Mizoguchi (2010) unterscheiden dabei **fünf zentrale Paradigmen**, die sich unter anderem in ihrer Granularität, diagnostischen Tiefe und Eignung für unterschiedliche Domänen unterscheiden:

1. **Cognitive Models:** Dieser Ansatz basiert auf der Idee, das Problemlöseverhalten von Lernenden durch eine detaillierte Abbildung kognitiver Prozesse nachzubilden. Häufig werden dazu regelbasierte Modelle genutzt, die in sogenannten *Model-Tracing*-Tutoren Anwendung finden. Diese Modelle bestehen aus Produktionsregeln, die typische Denk- und Handlungsschritte beim Lösen einer Aufgabe repräsentieren. Durch das Mitverfolgen der Bearbeitung können ITS auf dieser Basis Rückmeldungen geben, typische Fehler diagnostizieren oder passende Hilfen anbieten.

2. **Constraint-Based Models (CBM):** CBM verzichten auf die explizite Modellierung von Lösungswegen und konzentrieren sich stattdessen auf die Definition von *Constraints*, d. h. Bedingungen, die jede korrekte Lösung erfüllen muss. Der Fokus liegt auf der Analyse, ob die Lösung eines Lernenden gültige Zustände erzeugt. Dieser Ansatz ist besonders geeignet für offene oder kreative Aufgaben, bei denen viele verschiedene Lösungswege möglich sind. Ein Nachteil ist, dass CBMs keine Informationen über den konkreten Lösungsweg liefern und daher bei der Feedbackgenerierung eingeschränkt sind.

3. **Expert Systems:** Hier wird ein traditionelles Expertensystem in das ITS integriert, das entweder vollständige Lösungswege generiert oder als Referenzlösung dient. Lernendenantworten werden mit diesen Lösungen verglichen, um Fehler zu identifizieren oder gezielt Hinweise zu geben. Ein prominentes Beispiel ist GUIDON, das auf dem medizinischen Expertensystem MYCIN basiert („From Expert Systems to Intelligent Tutoring Systems" 1992). Diese Systeme sind oft stark domänenspezifisch und besonders geeignet für strukturierte Aufgabenbereiche mit klar definierbarem Expertenwissen.

4. **Partial Task Models:** Diese Methode zielt darauf ab, aus realen Nutzungsdaten häufige Bearbeitungsmuster zu extrahieren. Durch Data-Mining-Verfahren werden wiederkehrende Lösungsteile identifiziert, ohne dass ein vollständiges Aufgabenmodell nötig ist. Solche Modelle sind besonders für *ill-defined domains*[1] geeignet. Ihre Stärke liegt in der Flexibilität, ihre Schwäche in der fehlenden Generalisierbarkeit auf neue Aufgaben.

[1] unter *ill-defined Domains* versteht man Domänen, in denen u. a. Ziele nicht klar formuliert sind, Lösungswege uneindeutig und/oder zu denen keine vollständig formalen Theorien existieren

5. **Hybrid Approaches:** Da kein einzelner Ansatz universell geeignet ist, kombinieren viele moderne ITS mehrere der oben genannten Strategien. So kann etwa ein kognitives Modell zur Definition der Zielstruktur dienen, während gleichzeitig Daten aus partiellen Lösungswegen zur Verfeinerung des Feedbacks genutzt werden. Ein Beispiel ist der Canadarm-Tutor, der kognitive Modelle, Partial Task Models und automatische Vorschläge kombiniert (Fournier-Viger et al. 2013).

Diese Paradigmen verdeutlichen die Bandbreite möglicher Modellierungsstrategien in ITS und zeigen zugleich, dass kein einzelner Ansatz alle Anforderungen komplexer Lernprozesse abdecken kann. Während kognitive und constraint-basierte Modelle vor allem präzise Diagnose- und Regelstrukturen ermöglichen, bieten datengetriebene und hybride Ansätze größere Flexibilität im Umgang mit offenen, dynamischen Aufgaben.

2.3.3.3 ITS in der Programmierausbildung

Auch in der Programmierausbildung haben Intelligente Tutoring-Systeme Einzug gehalten. In diesem spezifischen Anwendungsfeld werden sie u. a. als *Intelligent Programming Tutors* (IPTs) bezeichnet (Crow, Luxton-Reilly und Wuensche 2018). Dabei unterscheidet sich dieser Einsatzbereich in mehrfacher Hinsicht von allgemeinen ITS-Szenarien: Programmieraufgaben erfordern nicht nur konzeptuelles Verständnis, sondern auch die korrekte Umsetzung in syntaktisch und semantisch gültigem Quellcode. Diese Besonderheiten stellen besondere Anforderungen an die Diagnosemechanismen und Feedback-Strategien entsprechender Systeme.

Im Vergleich zu klassischen ITS fokussieren IPTs besonders auf die Analyse von Programmcode. Diese Analyse bietet ein hohes Potenzial für detaillierte Diagnosen: Anders als bei Freitextantworten liegt mit dem Code ein formal interpretierbares Artefakt vor, das systematisch auf Fehler, Strukturen oder Lösungsstrategien untersucht werden kann. Dies erlaubt eine differenzierte Rückmeldung auf mehreren Ebenen – etwa zur Syntax, Semantik oder algorithmischen Qualität des Codes. Einige Systeme bieten zusätzlich Visualisierungen, automatische Planvergleiche oder strukturbezogene Hinweise an, um Lernende bei der Entwicklung korrekter Lösungsansätze zu unterstützen.

Le (2016) unterscheidet in einer systematischen Analyse von Programmierlernsystemen fünf spezifische *Feedback-Typen*, die in unterschiedlichen Kombinationen und Ausprägungen zur Anwendung kommen:

- **Syntax-Feedback:** Rückmeldungen, die auf der Compileranalyse basieren und häufig durch erklärende Hinweise zu typischen Syntaxfehlern ergänzt werden.
- **Semantisches Feedback:** Hinweise auf Basis der Analyse des Programmverhaltens oder Lösungswegs; teilweise kombiniert mit Ansätzen der Intentionserkennung.
- **Layout-Feedback:** Rückmeldungen zur Struktur, Lesbarkeit oder Einhaltung von Codekonventionen (z. B. Einrückungen, Namensgebung).
- **Qualitätsfeedback:** Bewertungen hinsichtlich Effizienz, Eleganz oder Ressourcennutzung einer Lösung.
- **Bestätigungsfeedback (Yes/No):** Binäre Rückmeldungen (richtig/falsch), ggf. ergänzt durch die Anzeige der korrekten Lösung.

Die Typologie ergänzt die in Tabelle 2.1 dargestellte Klassifikation von Keuning, Jeuring und Heeren (2019) um eine domänenspezifische, inhaltlich-diagnostische Perspektive. Während Keuning, Jeuring und Heeren Feedback inhaltlich nach seiner didaktischen Funktion differenzieren (z. B. konzeptbezogene Erklärungen, fehlerbezogene Diagnosen, prozedurale Hinweise), ordnet

Le (2016) Feedback entlang der für Programmieraufgaben typischen *Analyse- und Repräsentationsebenen* (Syntax, Semantik, Layout, Qualität, Bestätigung). Beide Ansätze sind somit komplementär: Keuning bietet eine didaktisch-funktionale Typisierung der *Art* der Rückmeldung, Le eine domänenspezifische Typisierung des *Gegenstands* bzw. der *Ebene*, auf die sich die Rückmeldung bezieht. In Kombination entsteht ein konsistenter Bezugsrahmen, der sowohl die pädagogische Funktion als auch die programmierdomänenspezifische Ziel- und Analyseebene von Feedback abdeckt. Während Le (2016) adaptive Rückmeldungen primär im Programmierkontext klassifiziert, betonen auch Crow, Luxton-Reilly und Wuensche (2018), dass ITS je nach Zielsetzung unterschiedliche Feedback-Strategien umsetzen: Sie reichen von automatisierter Fehlererkennung bis hin zu dialogisch-interaktiven Unterstützungsformen. Die zugrunde liegenden Diagnosemechanismen können dabei regelbasiert oder datengetrieben ausgestaltet sein (z. B. mittels Machine-Learning-Ansätzen), stets mit dem Ziel, Lernprozesse individuell und kontextsensitiv zu unterstützen.

Gleichwohl bestehen spezifische Herausforderungen: Die Vielfalt möglicher Lösungen bei offenen Programmieraufgaben erschwert die Entwicklung universell einsetzbarer Feedbackmechanismen. Viele IPTs beschränken sich daher entweder auf standardisierte Aufgaben oder auf begrenzte Teilaspekte wie die Korrektur von Syntaxfehlern oder die Diagnose bestimmter Fehlermuster. Nur wenige Systeme integrieren umfassende Diagnoseverfahren, die sowohl Programmcode als auch begleitende Planungs- oder Verständnisprozesse berücksichtigen. Crow, Luxton-Reilly und Wuensche (2018) zeigen in ihrer systematischen Übersicht, dass bestehende IPTs stark in ihren Schwerpunkten variieren: Manche Systeme fokussieren auf die Generierung von Hinweisen und Rückmeldungen innerhalb von Programmieraufgaben, andere auf die adaptive Navigation durch Aufgaben oder Lernmaterialien. Trotz dieser Fortschritte bleiben viele IPTs in ihrer Funktionalität begrenzt. Zahlreiche Systeme beschränken sich auf isolierte Aspekte des Lernprozesses und vernachlässigen etwa konzeptuelles Vorwissen, individuelle Lernstrategien oder alternative Lösungswege. Auch sind umfassende studentische Modellierungsansätze, wie sie für adaptive Lernunterstützung essenziell wären, bislang eher die Ausnahme als die Regel. Ein weiteres Defizit betrifft die mangelnde Integration didaktischer Erkenntnisse in die Gestaltung von Rückmeldungen – ein Problem, das in vielen ITS-Anwendungsfeldern beobachtet werden kann.

Insgesamt zeigt sich: Das Programmierenlernen ist ein vielversprechendes Einsatzfeld für ITS, bietet aber auch komplexe Herausforderungen. Die Möglichkeit, von Lernenden erzeugte Artefakte wie Quellcode maschinell auszuwerten, erlaubt präzise Diagnosen und differenzierte Feedback-Strategien. Gleichzeitig erfordert die hohe Variabilität von Lösungswegen, dass ITS in der Lage sind, nicht nur Fehler zu erkennen, sondern auch das zugrundeliegende Verständnismodell zu rekonstruieren – ein Ziel, das bislang nur teilweise erreicht wird.

Gerade vor dem Hintergrund dieser Grenzen gewinnen neuere Technologien – insbesondere aus dem Bereich der sog. *Künstlichen Intelligenz* – zunehmend an Bedeutung. Die Idee, das Verhalten menschlicher Tutorinnen und Tutoren algorithmisch zu simulieren, war dabei von Beginn an eng mit den jeweils aktuellen Entwicklungen in der KI-Forschung verknüpft. Bereits frühe ITS-Modelle nutzten wissensbasierte Inferenzverfahren, wie sie in der symbolischen KI der 1980er-Jahre verbreitet waren, während neuere Ansätze zunehmend auf datengetriebene, subsymbolische Verfahren zurückgreifen (Psotka, Massey und Mutter 1988; Crow, Luxton-Reilly und Wuensche 2018).

2.4 Künstliche Intelligenz

Bestrebungen, menschliches Verhalten durch Maschinen zu simulieren, reichen bis in die Antike zurück. In der griechischen Mythologie etwa konstruierten Hephaistos und Daidalos mechanische Wesen wie den bronzenen Wächter `Talos` oder sprechende Statuen (Devecka 2013; Mayor 2019). Diese frühen Vorstellungen und Artefakte zielten darauf ab, mechanische Abläufe zu gestalten, die menschliche Eigenschaften oder Handlungen nachahmen konnten. Sie zeugen von einer langen kulturellen Faszination, menschliche Fähigkeiten technisch zu reproduzieren.

Mit dem Aufkommen der Informatik als eigenständige Wissenschaft im 20. Jahrhundert verschob sich der Fokus: An die Stelle mechanischer Nachbildungen traten digitale Maschinen, die nicht nur Bewegungen, sondern auch geistige Prozesse simulieren sollten. Programmierbare Rechner eröffneten die Möglichkeit, Konzepte wie Lernen, Problemlösen oder Sprachverstehen algorithmisch zu modellieren. Einen entscheidenden Impuls erhielt diese Entwicklung 1956 durch das sogenannte Dartmouth Proposal, in dem John McCarthy gemeinsam mit Kollegen den Begriff *Artificial Intelligence* einführte. Darin formulierte er das ambitionierte Ziel, *dass jeder Aspekt des Lernens oder jede andere Form von Intelligenz so präzise beschrieben werden kann, dass eine Maschine sie simulieren kann* (McCarthy et al. 2006). Dieses Proposal gilt als Gründungsdokument und führte zur Dartmouth Conference, auf der die KI-Forschung als eigenständige wissenschaftliche Disziplin etabliert wurde.

Im deutschsprachigen Raum wurde der Begriff mit *Künstliche Intelligenz* (kurz: KI) übersetzt und hat sich in dieser Form etabliert. Lämmel (2020) weisen auf eine potenziell ungünstig gewählte Übersetzung des Begriffs hin, da diese Assoziationen zu einer übermächtigen künstlichen Entität wecke. Begriffe wie *maschinelle Intelligenz*, *synthetische Intelligenz* oder *gekünstelte Intelligenz* hätten aus ihrer Sicht möglicherweise den technischen Gehalt des Fachgebiets sachlicher wiedergegeben. Da sich der Begriff jedoch etabliert hat, erscheint eine präzise Verwendung und kontextbezogene Einordnung umso wichtiger.

Heute ist der Begriff weithin verbreitet und wird insbesondere im Kontext neuer digitaler Produkte und Dienstleistungen gerne als Schlagwort eingesetzt. Welche Technologien oder Verfahren damit konkret gemeint sind, bleibt jedoch meist intransparent. Bei genauerer Betrachtung – sofern möglich – stellt sich oft heraus, dass hinter den vermeintlich „intelligenten" Systemen klassische regelbasierte Entscheidungslogik oder einfache statistische Verfahren stehen, welche schon lange in der Informatik etabliert sind. Diese unscharfe Verwendung erschwert eine sachliche Auseinandersetzung und begünstigt zugleich unrealistische Erwartungen. Im Folgenden wird daher der Terminus *Künstliche Intelligenz* näher bestimmt und es werden die grundlegenden Paradigmen sowie methodischen Ansätze des Forschungsfeldes vorgestellt.

2.4.1 Definition

Eine einheitliche Definition von *Künstlicher Intelligenz* existiert nicht: Je nach disziplinärem Zugang, historischem Kontext und Anwendungsbereich variieren die Verständnisse und Zielsetzungen teils erheblich. Während manche Definitionen eng gefasst sind und sich etwa auf Verfahren des maschinellen Lernens beschränken, wird der Begriff in anderen Kontexten weit gefasst und umfasst sämtliche rechnergestützten Versuche, Aspekte menschlicher Intelligenz nachzubilden (Saghiri et al. 2022).

Ein häufig zitierter Definitionsvorschlag stammt von Elaine Rich, die *Künstliche Intelligenz* als "the study of how to make computers do things which, at the moment, people do better" beschreibt (Rich 1988) – eine Formulierung, die die technologische Dynamik und das normative Anspruchsniveau des Fachgebiets zugleich betont. Dieser Definitionsvorschlag macht deutlich, dass KI nicht nur eine technische Kategorie ist, sondern ein wandelbarer Bezugsrahmen, dessen

Gehalt sich mit technischen und gesellschaftlichen Entwicklungen verschiebt. Das besondere an dieser Definition ist, dass sie ohne den Begriff der Intelligenz auskommt und stattdessen den Fokus auf die Nachahmung menschlicher Fähigkeiten legt.

Die Ausführungen zeigen: Was heute als „KI" gilt, kann morgen bereits durch andere Verfahren ersetzt oder technologisch überholt sein. Der Begriff ist weder statisch noch eindeutig – seine inhaltliche Bedeutung verändert sich kontinuierlich im Zusammenspiel mit technischen Innovationen, gesellschaftlichen Diskursen und disziplinären Perspektiven. Eine reflektierte, kontextgebundene Verwendung ist daher unerlässlich.

Vor diesem Hintergrund wird in der vorliegenden Dissertation bewusst darauf verzichtet, den Begriff *Künstliche Intelligenz* als pauschales Label zu verwenden. Stattdessen werden konkrete Technologien, Paradigmen und Verfahren explizit benannt. Diese terminologische Präzision soll dazu beitragen, die technische Funktionsweise, didaktische Relevanz und potenzielle Limitationen einzelner Ansätze differenziert darzustellen. Wenn im Folgenden dennoch von *Künstlicher Intelligenz* die Rede ist, so geschieht dies als übergeordnete Bezeichnung für alle Bestrebungen, maschinell gestützte Systeme zu entwickeln, die Aspekte menschlichen Denkens, Entscheidens oder Handelns nachbilden. Die konkrete Analyse erfolgt jedoch stets anhand der jeweils eingesetzten Technologien.

2.4.2 Paradigmen Künstlicher Intelligenz: Wissensbasiert vs. datenbasiert

Vor dem Hintergrund der begrifflichen Unschärfe und des dynamischen Wandels von KI-Systemen erscheint es notwendig, zentrale technische Paradigmen zu differenzieren. Denn obwohl der Begriff *Künstliche Intelligenz* häufig als einheitliche Kategorie verwendet wird, existieren grundlegende Unterschiede in der Art und Weise, wie Systeme technisch modelliert und implementiert werden.

Im Rahmen dieser Arbeit werden zwei grundlegende Paradigmen unterschieden, die auch die weitere begriffliche Grundlage bilden: *wissensbasierte* (auch: *regelbasierte*) Ansätze und *datenbasierte* Ansätze. Erstere beruhen auf expliziter Repräsentation und Verarbeitung von Wissen, während letztere auf statistischem Lernen aus Beispielen und Interaktionen basieren. Beide Paradigmen haben spezifische Stärken und Grenzen, die in den folgenden Abschnitten detaillierter dargestellt werden. Zudem werden moderne hybride Architekturen berücksichtigt, die Elemente beider Ansätze kombinieren, um Transparenz und Steuerbarkeit einerseits sowie Flexibilität und Adaptivität andererseits zu verbinden.

Eine umfassende Darstellung aller KI-Technologien ist im Rahmen dieser Arbeit nicht möglich. Vielmehr werden die für den Kontext adaptiver Lernsysteme und kontextsensitiver Rückmeldungen besonders relevanten Ansätze und Begriffe eingeordnet. Für eine vertiefende Auseinandersetzung sei auf die einschlägige Literatur verwiesen, etwa Russell und Norvig (2023).

2.4.2.1 Wissensbasierte Ansätze

Wissensbasierte Ansätze beruhen auf der expliziten Modellierung von Wissen. Regeln, Fakten oder semantische Relationen werden dabei formalisiert und maschinenlesbar kodiert, um auf dieser Basis deterministische Entscheidungsprozesse zu ermöglichen. Zum Einsatz kommen logische Inferenzmechanismen, die auf festgelegten Wissensbasen operieren und somit transparente, nachvollziehbare Systemverhalten erlauben (Von Rueden et al. 2021; Russell und Norvig 2023).

Im Kontext von Lernsystemen – insbesondere bei der automatisierten Bereitstellung von Feedback – zeichnen sich wissensbasierte Verfahren durch ihre hohe Steuerbarkeit und interpretative Klarheit aus. Rückmeldungen lassen sich präzise definieren, etwa nach dem Muster: *Wenn Fehler*

X erkannt wird, dann gib Feedback Y. Solche Systeme beruhen typischerweise auf vordefinierten Fehlerkatalogen, Regelwerken oder diagnostischen Heuristiken, die eng mit den jeweiligen Aufgabenformaten verknüpft sind.

Ein klassisches Beispiel im Kontext der Programmierausbildung ist die Verwendung von *Bug Libraries* oder Fehlermusterdatenbanken: Wiederkehrende Syntax- oder Semantikfehler werden systematisch erfasst und mit spezifischen Rückmeldungen verknüpft (Le 2016). Auch in frühen Intelligenten Tutoring-Systemen – etwa dem *LISP Tutor* – kamen solche regelgeleiteten Strukturen zum Einsatz, um kontextspezifische Hinweise zu generieren. Die zugrunde liegende Feedbacklogik war dabei vollständig domänenspezifisch modelliert und formal operationalisiert (Anderson et al. 1990).

Besondere Stärken entfalten wissensbasierte Systeme in strukturierbaren Problemszenarien mit klar definierbaren Zielzuständen – etwa bei geschlossenen Aufgabenformaten oder festgelegten Lösungsschritten. Ihre Grenzen zeigen sich hingegen bei offenen Antwortformaten, unerwarteten Lösungswegen oder hoher semantischer Varianz in den Lernendenantworten. In solchen Fällen kann die Abdeckung potenzieller Fehlerzustände lückenhaft bleiben und die Generierung kontextsensitiver Rückmeldungen wird erschwert.

Trotz dieser Einschränkungen bleiben wissensbasierte Verfahren ein zentraler Bestandteil moderner hybrider Systemarchitekturen (Kierner, Kucharski und Kierner 2023; Van Bekkum et al. 2021; Singer et al. 2023). Sie können etwa als Entscheidungsstruktur zur Steuerung von Feedbacktypen dienen oder als semantische Filter bei der Verarbeitung von LLM-generierten Rückmeldungen eingesetzt werden. Ihre besondere Stärke liegt in der expliziten Modellierung didaktischer Intentionen und Regeln – ein Aspekt, der insbesondere im Kontext von Nachvollziehbarkeit und systematischer Steuerung von Lernprozessen weiterhin von hoher Relevanz ist.

2.4.2.2 Datenbasierte Ansätze

Datenbasierte Ansätze verfolgen einen konträren Zugang zu wissensbasierten Systemen: Anstatt Wissen explizit zu modellieren, werden Muster, Strukturen und Entscheidungsprozesse aus Daten „gelernt". Diese Paradigmen, häufig auch als *subsymbolisch* bezeichnet, basieren auf statistischen Methoden und Verfahren des maschinellen Lernens (ML). Die Systeme generalisieren aus Beispielen, anstatt auf vordefinierte Regelwerke zurückzugreifen (Russell und Norvig 2023).

Im Kontext von Lernsystemen zielen datenbasierte Ansätze darauf ab, aus großen Mengen an Nutzungs-, Interaktions- oder Lösungsdaten statistische Modelle abzuleiten. Diese Modelle können beispielsweise das aktuelle Kompetenzniveau eines Lernenden schätzen, typische Fehler vorhersagen oder kontextabhängige Rückmeldungen generieren. Bekannte Verfahren sind hier Entscheidungsbäume oder Bayes-Netze (Culbertson 2016; Gamboa und Fred 2001). Ein prominentes Beispiel aus der Forschung ist das *Bayesian Knowledge Tracing (BKT)*. Dieses Modell ermittelt auf probabilistischer Basis, wie wahrscheinlich es ist, dass ein Lernender ein bestimmtes Konzept bereits beherrscht (Corbett und Anderson 1995; Baker, Corbett und Aleven 2008). Rückmeldungen werden daraufhin datenbasiert angepasst – etwa durch die Auswahl geeigneter Aufgaben oder Hinweise zum weiteren Vorgehen. In neueren Lernumgebungen kommen zunehmend komplexere ML-Modelle zum Einsatz, die durch kontinuierliches Training auch mit heterogenen und offenen Antwortformaten umgehen können (Bernius, Krusche und Bruegge 2022).

Besondere Stärken datenbasierter Systeme liegen in ihrer Flexibilität und ihrer Fähigkeit, auch in unstrukturierten oder dynamischen Kontexten zu adaptieren. Sie benötigen keine vollständige Aufgabenmodellierung im Vorfeld, sondern „lernen" aus vergangenen Interaktionen. Das eröffnet insbesondere bei offenen Lernformaten – etwa bei Freitext- oder Programmieraufgaben – neue Möglichkeiten für kontextadaptive Feedbackprozesse.

Gleichzeitig bringen datenbasierte Ansätze auch einige Nachteile mit sich: Die Qualität des Outputs hängt stark von der Qualität und Repräsentativität der zugrunde liegenden Trainingsdaten ab, wodurch Verzerrungen oder fehlerhafte Generalisierungen entstehen können (Alzubaidi et al. 2021). Zudem ist die Nachvollziehbarkeit der Entscheidungen – insbesondere bei komplexen Modellen wie etwa tiefen neuronalen Netzen – häufig eingeschränkt, da sich interne Entscheidungsprozesse des Modells nur schwer interpretieren und erklären lassen (Islam et al. 2025; Alzubaidi et al. 2021). Für didaktische Anwendungen stellt dies ein relevantes Problem dar, da nicht nur die Korrektheit, sondern auch die Transparenz und Erklärbarkeit von Rückmeldungen entscheidend ist. Ein Ansatz in modernen Lernsystemen ist deshalb datenbasierte Verfahren mit wissensbasierten Komponenten zu kombinieren, um deren jeweilige Stärken zu bündeln (Von Rueden et al. 2021). Dies entspricht auch dem Ansatz der vorliegenden Arbeit, in der sowohl regelbasierte als auch datenbasierte Verfahren zur Generierung kontextsensitiver Rückmeldungen untersucht werden.

2.4.3 Generative Systeme

Mit der Veröffentlichung von ChatGPT im November 2022 begann eine neue Phase der breiten öffentlichen Aufmerksamkeit für Künstliche Intelligenz. Besonders **generative Systeme** rückten dabei in den Fokus – Modelle, die „neue" digitale Artefakte erzeugen können – sei es in Form von Texten, Bildern, Musik, Code oder anderen Medien. Bekannte Beispiele sind Text-to-Image-Modelle, Textgeneratoren wie `GPT-4` oder generative Musikmodelle wie `Jukebox` (Dhariwal et al. 2020; OpenAI et al. 2024). Ihnen gemeinsam ist, dass sie nicht nur auf bestehende Daten zurückgreifen, sondern Inhalte generieren, die statistisch den Trainingsdaten ähneln.

Eine zentrale Unterkategorie stellen dabei *Large Language Models* (LLMs) dar, die auf der von Vaswani et al. (2017) eingeführten *Transformer*-Architektur basieren und für die Generierung natürlicher Sprache entwickelt wurden (Vaswani et al. 2017; Howard und Ruder 2018). Sie werden auf sehr großen, heterogenen Datensätzen trainiert, die nicht nur Texte, sondern teilweise auch Code, Bilder oder multimodale Inhalte umfassen können. Ziel ist es, Wahrscheinlichkeitsverteilungen über Sequenzen zu modellieren und so kohärente Texte oder Dialoge zu erzeugen. Vor diesem Hintergrund hat sich die Bezeichnung *GPT* etabliert, die für *Generative Pre-trained Transformer* steht und die zentralen Eigenschaften dieser Modelle zusammenfasst: Sie sind (1) *generativ*, da sie neue Texte hervorbringen können (2) *pre-trained*, weil sie zunächst auf umfangreichen Korpora trainiert und anschließend für spezifische Aufgaben angepasst werden und (3) auf der Transformer-Architektur beruhen (Radford et al. 2018). Damit bilden GPT-Modelle heute die zentrale technologische Grundlage vieler generativer Systeme, insbesondere für Chatbots wie *ChatGPT*. Damit solche Systeme für den Dialogeinsatz geeignet sind, werden sie nach dem initialen Training durch Verfahren wie *Reinforcement Learning from Human Feedback* (RLHF) weiter angepasst (Christiano et al. 2017). Hierbei bewerten und annotieren Menschen Modellantworten, sodass die Ausgaben konsistenter, „hilfreicher" und näher an menschlicher Kommunikation wirken. Dieses Fine-Tuning verstärkt den Eindruck von Verständigkeit und Intentionalität – ohne dass tatsächlich kognitive Fähigkeiten des Systems vorliegen. Darüber hinaus ist davon auszugehen, dass bei vielen Anwendungen Filter- oder Moderationsschichten vorgeschaltet sind, um unerwünschte Eingaben oder Ausgaben abzufangen – etwa durch Blocklisten, Regex-Filter, Klassifikatoren oder systematische Guardrails (Rebedea et al. 2023). Da die genaue Architektur proprietärer Systeme wie ChatGPT jedoch nicht offengelegt wird, bleibt unklar, in welchem Umfang solche hybriden Komponenten eingesetzt werden.

Neben den technischen Aspekten gilt es insbesondere auch die sozialen und ethischen Implikationen dieser Systeme in den Blick zu nehmen. Das Labeln der Trainingsdaten sowie die Erstellung von Rückmeldungen für RLHF erfolgt häufig unter prekären Bedingungen: Recherchen zeigen, dass etwa kenianische Arbeitskräfte weniger als 2 USD pro Stunde erhielten, während sie Inhalte

mit gewalt- oder missbrauchsbezogenem Inhalt annotierten und darunter psychisch litten (Perrigo 2023). Zugleich bleibt die Datengrundlage der Modelle intransparent, sodass Fragen nach Urheberrechten, Verzerrungen (sog. *Bias*) und Fairness bestehen.

Ein weiteres Problemfeld ist die Tendenz zur Anthropomorphisierung. Viele Chatbots nutzen Formulierungen oder Interface-Elemente, die suggerieren, dass das System „nachdenke" oder über eigene Absichten verfüge. Tatsächlich handelt es sich jedoch um statistische Mustererkennung und Wahrscheinlichkeitsberechnung, nicht um menschliches Denken oder Bewusstsein. Diese Fehlwahrnehmung birgt das Risiko, dass Nutzerinnen und Nutzer die Fähigkeiten der Systeme überschätzen, falsches Vertrauen entwickeln oder die technologischen Grenzen nicht erkennen (Hasan et al. 2025; Maeda 2025; Marchegiani 2025).

Generative Systeme lassen sich somit weder eindeutig den klassischen wissens- noch den datenbasierten Ansätzen zuordnen. Sie knüpfen zwar an datenbasierte Verfahren an, unterscheiden sich jedoch von früheren Ansätzen durch ihre enorme Skalierung, ihre Architektur und ihre generative Ausrichtung (Yazdani et al. 2025). Ihre Potenziale für Anwendungen in Lernsystemen – etwa bei der Generierung von Feedback oder der Simulation von Dialogen – stehen in einem Spannungsfeld mit erheblichen Herausforderungen: mangelnde Transparenz, ethische Bedenken, Risiken der Fehlnutzung und die Gefahr der Fehlinterpretation durch Anthropomorphisierung. Demgegenüber stehen Chancen, Lernprozesse stärker zu individualisieren, gezielt zu unterstützen und zu erweitern. Gerade in Domänen wie der Programmierung, die auf symbolischen Repräsentationen und formalen Strukturen beruht und für die große Datenmengen beim Training verfügbar sind, zeigen sich bereits heute vielversprechende Einsatzmöglichkeiten (Yan et al. 2025; MacNeil et al. 2022b; MacNeil et al. 2022a; Denny et al. 2021).

Kapitel 3

Forschungsrahmen und Forschungsfragen

Die bisherigen Ausführungen haben gezeigt: Feedback ist ein vielschichtiges Konstrukt, das sich entlang verschiedener Dimensionen – etwa Quelle, Inhalt, Form, Zeitpunkt und Funktion – differenzieren lässt (siehe Abschnitt 2.1.2). Der Forschungsstand verdeutlicht zugleich, dass gerade im Kontext des Programmierlernens zentrale Fragen bislang unbeantwortet bleiben. Zwar existieren zahlreiche Studien zu allgemeinen Wirkmechanismen von Feedback, jedoch fehlt es an systematischen Ansätzen, diese Erkenntnisse auf die spezifischen Anforderungen des Programmierenlernens zu übertragen und für digitale Lernumgebungen nutzbar zu machen.

Besonders auffällig ist, dass bestehende Arbeiten überwiegend Einzelaspekte beleuchten, während eine integrative Perspektive auf *Feedback-Strategien* fehlt. Ziel dieser Arbeit ist es, solche Strategien so zu beschreiben und zu formalisieren, dass sie einerseits empirisch begründet, andererseits technologisch operationalisierbar sind.

Aus dieser Analyse ergeben sich mehrere Forschungslücken, die den Rahmen der vorliegenden Arbeit bestimmen. Im Folgenden werden diese Desiderata systematisiert, bevor daraus die übergeordnete Forschungsfrage und die einzelnen Teilfragen abgeleitet werden.

3.1 Forschungsdesiderata

Es fehlt an empirisch fundiertem Wissen darüber, welche *Feedback-Strategien* im Kontext der Programmierausbildung unter welchen Bedingungen wirksam sind. Unklar ist insbesondere, welche Indikatoren geeignet sind, um Eingriffe in den Programmierprozess auszulösen, welche Arten von Feedback in welchen Situationen wirksam sind und wie individuelle sowie situative Faktoren die Effektivität dieser Strategien beeinflussen. Um die Wirksamkeit von Feedback-Strategien erforschen zu können, bedarf es zunächst einer systematischen Erfassung und Analyse der realen Feedbackpraxis von Lehrpersonen in der Programmierausbildung, um daraus empirisch fundierte Gestaltungsprinzipien abzuleiten.

Bislang existiert kein formalisiertes Modell, das Aufgabenstrukturen, Lernendenantworten und den relevanten Kontext so beschreibt, dass darauf aufbauend Feedback-Strategien systematisch abgeleitet und algorithmisch umgesetzt werden können. Eine solche Formalisierung ist jedoch notwendig, um kontextsensitive Rückmeldungen in digitalen Lernumgebungen nachvollziehbar, konsistent und adaptiv zu gestalten.

Zwar zeigen erste Studien, dass große Sprachmodelle grundsätzlich Potenziale für die Generierung von Feedback besitzen, unklar bleibt jedoch, in welchem Maße sie in der Lage sind, *kontextsensitive* Rückmeldungen valide bereitzustellen. Insbesondere ist offen, inwieweit unterschiedliche inhaltliche Feedback-Typen, wie sie in Abschnitt 2.1.2 dargestellt wurden, durch die Bereitstellung ausreichenden Kontexts zuverlässig erzeugt werden können.

Zwar existieren bereits Lernumgebungen, die regelbasierte und datenbasierte Ansätze zur Unterstützung von Lernprozessen kombinieren; jedoch fehlt ein systematischer Zugang, der auf empirisch fundierten Erkenntnissen zu Feedback-Strategien basiert und diese in skalierbarer und zugleich nachvollziehbarer Weise technisch umsetzbar macht. Besonders relevant ist hierbei die Frage, wie sich regelbasierte Verfahren – etwa zur Operationalisierung didaktischer Prinzipien – und datenbasierte Verfahren – wie Machine Learning oder LLMs – so integrieren lassen, dass sie die Entwicklung und Anwendung kontextsensitiver Feedback-Strategien wirksam unterstützen.

3.2 Forschungsfragen

Aus den zuvor beschriebenen theoretischen und empirischen Desiderata ergibt sich die leitende Forschungsfrage dieser Dissertation:

> **Wie können Feedback-Strategien modelliert und formalisiert werden, damit kontextsensitives Feedback in Programmierlernsystemen nachvollziehbar, skalierbar und systematisch erforschbar umgesetzt werden kann?**

Diese Leitfrage zielt auf die Verbindung dreier bislang weitgehend isolierter Perspektiven:

(1) der *empirischen* Analyse realer Feedbackpraktiken,

(2) der *formalen* Modellierung ihrer zugrunde liegenden Strukturen und

(3) der *technologischen* Umsetzung in digitalen Lernumgebungen.

Damit wird ein Brückenschlag zwischen didaktischer Theorie, empirischer Beobachtung und technischer Realisierung angestrebt.

Struktur und Logik der Forschungsfragen

Die Ableitung der Teilfragen folgt einer schrittweisen, methodisch aufeinander bezogenen Progression: Ausgehend von der Analyse realer Feedbackpraxis (Befundebene) werden in einem zweiten Schritt formale Modelle entwickelt (Modellebene), die schließlich in technologischen Umsetzungen überprüft werden (Implementierungsebene). Diese drei Ebenen sind im Sinne einer *Triangulation* konzipiert: Empirie informiert die Modellierung, die Modellierung schafft prüfbare Artefakte, und die Artefakte liefern Rückkopplungen für empirisch-theoretische Präzisierungen. Die Arbeit integriert somit drei komplementäre Erkenntnisperspektiven:

1. **Empirische Ebene:** Untersuchung, wie erfahrene Lehrpersonen Feedbackentscheidungen treffen und welche Formen ihre Rückmeldungen in realen Unterrichtssituationen annehmen.
2. **Formale Ebene:** Entwicklung eines Bezugsmodells zur Beschreibung von Aufgaben, Antworten und Kontextbedingungen, das als Grundlage für algorithmische Feedbackprozesse dient.
3. **Technologische Ebene:** Erprobung generativer und regelbasierter Verfahren zur Umsetzung kontextsensitiven Feedbacks in digitalen Lernumgebungen.

Die folgenden Forschungsfragen konkretisieren diese Ebenen und strukturieren zugleich den Aufbau der Arbeit. Ausgangspunkt bildet die Frage, wie Lehrpersonen im realen Lehr-Lernkontext Feedbackentscheidungen treffen. Ziel ist es, Entscheidungslogiken und qualitative Merkmale ihrer Rückmeldungen zu identifizieren, um daraus didaktisch relevante Kategorien für adaptive Systeme abzuleiten.

FF1: Welche Faktoren prägen Entscheidungsprozesse von Lehrpersonen zum Einsatz von Feedback in der Programmierausbildung und welche Charakteristika weisen die gegebenen Rückmeldungen in der Praxis auf? (Kapitel 4)

- FF1.1: Auf Grundlage welcher individueller und instruktionaler Faktoren entscheiden sich Lehrpersonen in der Programmierausbildung *für* oder *gegen* Feedback in Programmierprozessen?
- FF1.2: Welche inhaltlichen Ausprägungen zeigen die von Lehrpersonen formulierten Rückmeldungen im Rahmen von Programmierprozessen?

Die gewonnenen Antworten zu **FF1** liefern die empirische Grundlage für eine strukturierte Beschreibung des Gegenstandsbereichs. Um aus Einzelfällen generalisierbare und in digitalen Systemen operationalisierbare Gestaltungsprinzipien abzuleiten, bedarf es einer formalen Repräsentation der relevanten Objekte und Relationen (Aufgaben, Antworten, Kontext, Entscheidungsregeln).

FF2: Wie lassen sich Programmieraufgaben und Lernendenantworten formalisieren, sodass individuelle und instruktionale Faktoren systematisch abgebildet und als Grundlage für die automatisierte Bereitstellung kontextsensitiven Feedbacks in Lernplattformen genutzt werden können? (Kapitel 5)

- FF2.1: Welche zentralen Komponenten von Programmieraufgaben sind für die automatisierte Bereitstellung kontextsensitiven Feedbacks relevant und welche Ansätze zu ihrer Modellierung bzw. Formalisierung lassen sich identifizieren?
- FF2.2: Wie lassen sich Lernendenantworten zu Programmieraufgaben kategorisieren, sodass sie für die automatisierte Diagnose und die Bereitstellung kontextsensitiven Feedbacks genutzt werden können?
- FF2.3: Wie können die Dimensionen von Aufgaben und Lernendenantworten in einem integrativen Modell formal zusammengeführt und strukturiert werden, sodass sie für die automatisierte Bereitstellung kontextsensitiven Feedbacks nutzbar gemacht werden?

Auf Basis eines solchen formalen Bezugsrahmens lässt sich prüfen, inwiefern generative Verfahren in der Lage sind, aus explizit bereitgestelltem Kontext differenzierte, inhaltlich konsistente Rückmeldungen abzuleiten. Dies führt zur technologischen Bewertung von LLMs unter kontrollierter Kontextvariation.

FF3: Inwiefern eignen sich große Sprachmodelle zur Generierung kontextsensitiven Feedbacks für das Programmierenlernen? (Kapitel 6)

- FF3.1: Welche Charakteristika weisen von großen Sprachmodellen generierte Rückmeldungen auf und wie verändert sich deren Qualität in Abhängigkeit vom bereitgestellten Kontext?
- FF3.2: Inwieweit lassen sich große Sprachmodelle zur Generierung unterschiedlicher inhaltlicher Feedback-Typen einsetzen und welche Herausforderungen und Grenzen treten dabei auf?

Unabhängig von der konkreten Technologie verlangt die nachhaltige Nutzung kontextsensitiven Feedbacks eine tragfähige Systemperspektive. Gefragt sind Architekturen, die formale Modelle, regelbasierte Entscheidungen und datenbasierte Verfahren in nachvollziehbaren Datenflüssen verbinden und so den Betrieb in realen Lernumgebungen ermöglichen.

FF4: Welche architektonischen Prinzipien und Komponenten sind erforderlich, um empirisch fundierte und formal beschriebene Feedback-Strategien in digitalen Lernumgebungen für das Programmierlernen skalierbar und nachvollziehbar umzusetzen? (Kapitel 7)

3.3 Forschungsdesign

Das Forschungsdesign dieser Arbeit orientiert sich an Prinzipien des *Design-Science Research* (DSR) (Hevner und Chatterjee 2010; Stokes 1997). Ausgangspunkt ist ein praxisrelevantes Problemfeld, das empirisch untersucht, theoretisch modelliert und in technologischen Umsetzungen überprüft wird. Während klassisches DSR typischerweise mehrere Iterationen vorsieht, versteht sich die vorliegende Dissertation als erste exemplarische Iteration innerhalb eines *mehrphasigen und multidimensional ausgerichteten* Forschungsprogramms.

3.3.1 Design Science Research als Forschungsrahmen

Das Paradigma des *Design Science Research* (DSR) ist in der informationswissenschaftlichen und techniknahen Bildungsforschung als designorientierter Ansatz zu verorten (Hevner et al. 2004). Während die verhaltenswissenschaftliche Forschung (Behaviourismus) darauf abzielt, Theorien zu entwickeln und zu prüfen, die menschliches Handeln und organisationale Prozesse erklären oder vorhersagen, verfolgt DSR das Ziel, innovative Artefakte zu entwerfen und zu evaluieren, die menschliche und organisationale Fähigkeiten erweitern (Hevner et al. 2004; Hevner und Chatterjee 2010). Artefakte im Sinne von DSR können sehr unterschiedliche Formen annehmen – von konzeptuellen Modellen und Methoden über Algorithmen bis hin zu prototypischen Implementierungen.

Die methodische Abgrenzung zu anderen Forschungsansätzen wird besonders im Vergleich mit handlungsorientierten Verfahren wie der Aktionsforschung deutlich: Während Aktionsforschung auf Veränderung und gemeinsames Lernen im praktischen Feld ausgerichtet ist, fokussiert DSR auf die systematische Konstruktion und Evaluation von Artefakten, die zugleich wissenschaftlichen und praktischen Erkenntnisfortschritt ermöglichen (Iivari 2007).

Die Ursprünge von DSR liegen einerseits in den Ingenieurwissenschaften und andererseits in den sog. *Sciences of the Artificial* nach Simon (2008). In der Informatik und den Wirtschaftsinformationssystemen wurde DSR bereits früh praktiziert, etwa durch die Entwicklung neuer Programmiersprachen, Algorithmen und Systemarchitekturen (Iivari 2007). Als eigenständiges Paradigma wurde es jedoch erst seit den 1990er Jahren systematisch theoretisiert (March und Smith 1995; Hevner et al. 2004). Seither gilt DSR als komplementärer Ansatz zur verhaltenswissenschaftlichen Forschung und wird zunehmend als gleichwertige Säule der Informationssystemforschung anerkannt.

Das **Ziel** von DSR besteht darin, Probleme und Chancen aus einem realitätsnahen Anwendungskontext aufzugreifen, diesbezüglich innovative Artefakte zu entwickeln und deren Nutzen sowohl wissenschaftlich fundiert als auch praktisch überprüfbar zu machen. Im Unterschied zu rein technischer Systementwicklung ist DSR dadurch gekennzeichnet, dass Artefakte nicht nur konstruiert, sondern zugleich als Forschungsbeiträge theoretisch verankert und empirisch evaluiert werden. Besondere Bedeutung kommt dabei dem Kriterium der *Innovativität* zu. Ein Design gilt in DSR nur dann als wissenschaftlich relevant, wenn es über ein *Routine-Design* hinausgeht (Hevner 2007). Routine-Design bezeichnet die Anwendung bereits bekannter Prozesse, Methoden oder Artefakte auf neue Probleme. DSR hingegen fordert einen originären Beitrag, sei es in Form neuer Konzepte, veränderter Strukturen oder bislang unerprobter Verfahren. Nur dadurch wird gewährleistet, dass die Arbeit nicht lediglich bestehendes Wissen reproduziert, sondern die Wissensbasis substantiell erweitert.

Das Vorgehen beim DSR folgt typischerweise einem *Build-and-Evaluate*-Zyklus: Artefakte werden entworfen, implementiert und anhand geeigneter Methoden überprüft, wobei sich dieser Zyklus mehrfach wiederholen kann, bis ein zufriedenstellendes Ergebnis vorliegt (March und Smith 1995).

Zur systematischen Strukturierung des Forschungsprozesses schlägt Hevner (2007) ein Rahmenmodell vor, das **drei ineinandergreifende Zyklen** unterscheidet: den *Relevance Cycle*, den *Rigor Cycle* und den *Design Cycle*. Damit werden der praktische Anwendungskontext, die wissenschaftliche Wissensbasis und die iterative Artefaktentwicklung in einem integrierten Prozess miteinander verknüpft.

Im **Relevance Cycle** werden die Forschungsaktivitäten mit dem Anwendungskontext verknüpft. Anforderungen aus der Umwelt werden in den Forschungsprozess eingebettet, indem Probleme, Potenziale und Zielsetzungen identifiziert werden, die für die Entwicklung von Artefakten relevant sind. Gleichzeitig werden Akzeptanz- und Erfolgskriterien definiert, anhand derer die entwickelten Artefakte in Labor- oder Feldstudien evaluiert werden (Hevner 2007). Damit stellt der Relevance Cycle sicher, dass DSR einen erkennbaren Nutzen für die Praxis hat und an realen Herausforderungen ausgerichtet ist. In dieser Arbeit wird der Relevance Cycle durch die empirische Exploration realer Feedbackpraxis in der Programmierausbildung aktiviert (Kapitel 4). Die Untersuchung erfahrener Lehrpersonen liefert Anforderungen an kontextsensitive *Feedback-Strategien* (Entscheidungsanlässe, Kontextfaktoren, inhaltliche Ausprägungen) und definiert Erfolgskriterien für spätere Artefakte (z. B. Nachvollziehbarkeit, situative Passung, didaktische Angemessenheit). Forschungslogisch entspricht dies **FF1** und stellt die anwendungsnahe Problemdefinition bereit.

Im **Rigor Cycle** wird der Forschungsprozess mit der wissenschaftlichen Wissensbasis verbunden. Hier werden Theorien, Modelle, Methoden sowie Erfahrungswissen, die als Grundlage für die Konstruktion und Evaluation von Artefakten dienen, bereitgestellt. Zugleich werden hier die im Forschungsprozess gewonnenen Erkenntnisse in die Wissensbasis zurückgeführt und sie so um neue Konzepte, Modelle oder Designprinzipien erweitert (Iivari 2007). Auf diese Weise wird gewährleistet, dass die entwickelten Artefakte nicht bloß praktische Lösungen darstellen, sondern wissenschaftlich originäre Beiträge liefern. In dieser Arbeit werden die empirischen Befunde mit der einschlägigen Wissensbasis verbunden und in ein *formales* Rahmenmodell überführt (Kapitel 5). Kernelemente sind das Y-Modell, die Formalisierung von Aufgaben- und Antwortdimensionen sowie das Konzept der *Antwortklassen* als entscheidungs- und diagnostikfähige Repräsentationen. Methodisch werden dabei Kriterien wie *Objektivität*, *Entscheidbarkeit* und *Beobachtbarkeit* adressiert, um anschlussfähige, maschinenlesbare Strukturen für Feedback-Strategien zu schaffen. Dies entspricht **FF2** und erweitert die Wissensbasis um präzise, operationalisierbare Modellbausteine.

Im Zentrum des Rahmenmodells steht der **Design Cycle**, der den iterativen Prozess von Artefaktkonstruktion und -evaluation beschreibt. Dieser zeichnet sich durch Iterationen zwischen dem Entwurf von Designalternativen und deren systematischer Überprüfung aus. Ziel ist es, das Artefakt so lange zu verfeinern, bis es die im Relevance Cycle formulierten Anforderungen erfüllt und sich mit den im Rigor Cycle bereitgestellten theoretischen Grundlagen in Einklang bringen lässt (Hevner et al. 2004). In dieser Arbeit umfasst der Design Cycle (i) die kontrollierte *Technologieprüfung* generativer Modelle (Kapitel 6), in der die Fähigkeit von LLMs zur kontextsensitiven Erzeugung differenzierter Feedback-Typen unter expliziter Kontextvariation untersucht wird (**FF3.1**, **FF3.2**), sowie (ii) die *Systemperspektive* (Kapitel 7), in der mit `APFEL` eine referenzielle Architektur entwickelt wird, die regelbasierte und datengetriebene Komponenten integriert und Nachvollziehbarkeit, Skalierbarkeit und didaktische Steuerbarkeit als Leitkriterien implementiert (**FF4**).

Das Drei-Zyklen-Modell verdeutlicht die Interdependenz der drei Perspektiven. Der Relevance Cycle sichert die praktische Relevanz der Forschung, indem er Anforderungen und Bewertungsmaßstäbe aus der Umwelt in den Prozess einbringt. Der Rigor Cycle sorgt für wissenschaftliche Fundierung und garantiert, dass neue Artefakte als originäre Forschungsbeiträge in die Wissens-

basis eingehen. Der Design Cycle schließlich setzt diese beiden Ströme in konkrete Artefakte um, die in engen Iterationen entwickelt und getestet werden. Erst in der Verbindung dieser drei Zyklen wird DSR als eigenständiges Forschungsparadigma klar gegenüber rein praktischer Systementwicklung oder rein theoretischer Grundlagenforschung abgegrenzt. Gute Designforschung zeichnet sich nach (Hevner et al. 2004) daher erst durch eine ausgewogene Balance von Relevanz, Rigor und Design aus.

Im Sinne von *Design Science Research* (DSR) verknüpft diese Dissertation praxisrelevante Problemstellungen der Programmierausbildung mit theoretisch fundierter Artefaktentwicklung und deren Evaluation (Hevner et al. 2004; Hevner 2007; Hevner und Chatterjee 2010).

Konkret gliedert sich das Vorgehen in vier aufeinander aufbauende Schritte, die den empirischen, formalen und technologischen Ebenen der Forschungsfragen entsprechen:

(S1) **Von Praxis zu Empirie (Kapitel 4):** Untersuchung der Entscheidungsgrundlagen und Charakteristika von Feedback erfahrener Programmierlehrender (FF1.1, FF1.2). Grundlage ist eine Umfrage, in der Lehrende Rückmeldungen zu typischen Bearbeitungssequenzen von Lernenden geben. Die Daten werden mittels qualitativer Inhaltsanalyse nach Mayring ausgewertet. Ziel ist die Entwicklung eines Kategoriensystems, das Entscheidungsprozesse sowie inhaltliche Ausprägungen von Feedback systematisch erfasst. Dieses Kapitel erfüllt die Funktion der empirischen Exploration und Problemdefinition im DSR-Sinne.

(S2) **Von Empirie zu Theorie (Kapitel 5):** Aufbauend auf den empirischen Befunden und einschlägiger Literatur werden Aufgaben, Lernendenantworten und Kontextinformationen formalisiert (FF2). Ergebnis ist eine maschinenlesbare Repräsentation, die situative und individuelle Bedingungen ebenso wie inhaltliche Aspekte strukturiert erfasst. Damit werden die theoretisch-formalen Grundlagen für algorithmisch anschlussfähige Feedback-Strategien geschaffen.

(S3) **Von Theorie zu Technologie (Kapitel 6):** Im dritten Schritt wird ein technologisches Artefakt gestaltet und evaluiert. Es wird untersucht, inwiefern große Sprachmodelle (LLMs) zur Generierung kontextsensitiven Feedbacks geeignet sind (FF3.1, FF3.2). Dazu werden Rückmeldungen des LLM unter Variation des zur Verfügung gestellten Kontexts erhoben und qualitativ ausgewertet. Im Vordergrund steht die Frage, ob sich mit geeigneten Prompts differenzierte Feedback-Typen konsistent erzeugen lassen. Dieses Kapitel ist daher dem Design und der Evaluation eines Artefakts im Rahmen von DSR zuzuordnen.

(S4) **Von Technologie zu System (Kapitel 7):** Abschließend werden architektonische Prinzipien, Schnittstellen und Datenflüsse für digitale Lernumgebungen konzipiert. Die Referenzimplementierung dient der konzeptionellen Demonstration, wie formal beschriebene Feedback-Strategien skalierbar und nachvollziehbar in Systemarchitekturen überführt werden können. Die Bewertung erfolgt anhand von Anwendungsszenarien und Designrationalen.

3.3.2 Qualitative Inhaltsanalyse im Rahmen des Forschungsdesigns

Die qualitative Inhaltsanalyse ist im qualitativen Paradigma der empirischen Sozialforschung verortet. Sie richtet den Fokus nicht auf die Quantifizierung von Daten, sondern auf die systematische, regelgeleitete und theoriegeleitete Analyse von Texten und anderen Kommunikationsinhalten (Mayring 2015b). Anders als rein interpretative Verfahren strebt sie eine methodisch kontrollierte Auswertung an, die nachvollziehbar und intersubjektiv überprüfbar bleibt.

Die Ursprünge der Methode liegen in den 1970er- und 1980er-Jahren, als Philipp Mayring die klassischen quantitativen Verfahren der Inhaltsanalyse (etwa nach Berelson 1952) um qualitative

Zugänge erweiterte. Ziel war es, auch komplexere, inhaltlich tiefere Dimensionen von Textmaterial erschließen zu können, ohne dabei die Systematik der Inhaltsanalyse zu verlieren (Mayring 2015b). Heute wird die qualitative Inhaltsanalyse vielfältig angewendet, etwa in der Psychologie, Soziologie, Kommunikationswissenschaft und Erziehungswissenschaft. Sie dient vor allem dazu, latente Sinngehalte, Strukturen und Muster in Textmaterial herauszuarbeiten, Kategorien zu entwickeln und diese theoriegeleitet oder materialnah auf empirische Daten anzuwenden (Kuckartz 2018).

Die qualitative Inhaltsanalyse nach Mayring ist in der empirischen Bildungsforschung fest etabliert und gilt als ein methodisches Standardverfahren zur systematischen Auswertung textbasierter Daten (Gläser-Zikuda, Hagenauer und Stephan 2020; Göhner und Krell 2020). Gläser-Zikuda, Hagenauer und Stephan (2020) betonen insb. das Potenzial der Methode, komplexe Lern- und Lehrprozesse in ihrer Vielschichtigkeit zu erfassen und gleichzeitig methodische Gütekriterien zu sichern. In einem Review-Artikel zeigen Göhner und Krell (2020) darüber hinaus, dass die qualitative Inhaltsanalyse in den Naturwissenschaftsdidaktiken breit genutzt wird, wobei verschiedene Umsetzungsvarianten und der Umgang mit Gütekriterien systematisch diskutiert werden.

Die Einsatzmöglichkeiten sind vielfältig: von der Analyse von Interviews mit Lehrkräften über die Untersuchung von Lernmaterialien bis hin zur Betrachtung von schulischen Entwicklungsprozessen (Haberbosch et al. 2025; Homt und Bloh 2025). Dabei wird häufig eine Mischform aus deduktiver und induktiver Kategorienbildung realisiert, bei der theoriegeleitete Kategorien durch materialbasierte Erweiterungen ergänzt werden (Homt und Bloh 2025). Dies ermöglicht es, sowohl vorhandenes theoretisches Wissen einzubeziehen als auch neue Phänomene sichtbar zu machen. Zunehmend findet die Methode auch in der Programmierausbildung Anwendung. So wird sie genutzt, um Kompetenzmodelle im Bereich des Programmierens zu entwickeln und die Lernprozesse von Programmierlernenden systematisch zu erfassen (Kiesler 2020a; Kiesler 2020b). Hier zeigt sich, dass die qualitative Inhaltsanalyse nicht nur in klassischen schulischen Kontexten, sondern auch in technologiebezogenen Bildungsfeldern wichtige Einsichten liefern kann.

Darüber hinaus ist die Methode auch in der Feedback-Forschung verbreitet. Aktuelle Untersuchungen verwenden qualitative Inhaltsanalyse, um Rückmeldungen, welche durch LLMs generiert wurden, zu analysieren. So untersuchten Rüdian et al. (2025) die Wahrnehmung von Studierenden hinsichtlich Feedback von Lehrkräften im Vergleich zu Feedback von LLMs und nutzten dabei ein kodierendes Vorgehen nach Mayring (Rüdian et al. 2025). Ebenso verglich Baz (2025) Chatbot-basiertes Feedback mit Lehrer-Feedback zu Schülertexten und wertete beide Formen mithilfe qualitativer Inhaltsanalyse aus (Baz und Hasırcı Aksoy 2025).

3.3.2.1 Vorgehen und Techniken

Die qualitative Inhaltsanalyse zeichnet sich durch ein regelgeleitetes und transparent dokumentiertes Vorgehen aus. Ziel ist es, große Mengen qualitativen Datenmaterials in einer nachvollziehbaren Abfolge von Schritten zu reduzieren, zu strukturieren und theoriebezogen auszuwerten (Mayring 2015b).

Zentral ist dabei die Entwicklung eines Kategoriensystems, das den Kern der Analyse bildet. Kategorien können *induktiv* aus dem Material heraus entwickelt werden, indem wiederkehrende Themen, Muster oder Sinngehalte identifiziert und abstrahiert werden. Alternativ oder ergänzend können Kategorien *deduktiv* aus bestehenden Theorien, Modellen oder Forschungsfragen abgeleitet werden (Schreier 2012). In der Praxis wird häufig eine Mischform realisiert: Theoriegeleitete Kategorien dienen als Ausgangspunkt, die im Verlauf der Analyse durch induktive Ergänzungen und Präzisierungen erweitert werden. Dadurch bleibt das Verfahren sowohl offen für neue, im Material auftauchende Aspekte als auch eng mit theoretischen Vorannahmen verknüpft (Mayring 2015b).

Der Ablauf der qualitativen Inhaltsanalyse nach Mayring folgt typischerweise einem festen Schema. Zunächst werden das Analysematerial und die Analyseeinheiten festgelegt (z. B. Interviewpassagen, Textsegmente oder Dokumentenausschnitte). Danach werden die Kategorien definiert und mit Kodierregeln versehen, um eine einheitliche Anwendung zu gewährleisten. Anschließend erfolgt die eigentliche Kodierung des Materials, d. h. die Zuordnung von Textstellen zu Kategorien. Im Zuge dieser Kodierung kann das Kategoriensystem iterativ überprüft, überarbeitet und erweitert werden. Abschließend werden die Ergebnisse in einer zusammenfassenden Darstellung verdichtet und interpretiert (Mayring 2015b; Schreier 2012).

Ein wesentliches Merkmal der qualitativen Inhaltsanalyse ist somit die Kombination aus *systematischer Kategorienbildung*, *regelgeleitetem Vorgehen* und *inhaltlicher Interpretation*. Dadurch wird eine Balance zwischen Offenheit gegenüber dem Material und methodischer Strenge erreicht, die sowohl explorative als auch hypothesengeleitete Forschung erlaubt.

Versuche, dieses regelgeleitete Vorgehen durch den Einsatz von LLMs zu automatisieren, haben bislang nicht überzeugt. Mayring (2025) zeigt in einer systematischen Untersuchung, dass ChatGPT (Version 3.5 und 4) zwar in der Lage ist, grobe thematische Zusammenfassungen zu generieren, dabei jedoch zentrale Anforderungen der qualitativen Inhaltsanalyse verfehlt: Weder die präzise Kategorienbildung noch die Einhaltung methodischer Prozessschritte oder die Überprüfung von Kodierübereinstimmungen wurden zuverlässig umgesetzt. LLMs erscheinen damit höchstens als exploratives Hilfsmittel für erste thematische Übersichten geeignet, nicht jedoch als Ersatz für eine methodisch fundierte Inhaltsanalyse, wie sie im Rahmen dieser Arbeit durchgeführt wird.

3.3.2.2 Gütekriterien

Wie jedes empirische Verfahren muss auch die qualitative Inhaltsanalyse die Einhaltung wissenschaftlicher Gütekriterien sicherstellen. Da die klassischen Gütekriterien der quantitativen Forschung (Objektivität, Reliabilität, Validität) nicht ohne Weiteres auf qualitative Verfahren übertragbar sind, wurden diese an die Besonderheiten interpretativer Forschung angepasst (Mayring 2015b; Schreier 2012).

Ein zentrales Kriterium ist die *Transparenz des Forschungsprozesses*. Sämtliche Entscheidungen zur Materialauswahl, Kategorienbildung und Kodierung müssen nachvollziehbar dokumentiert werden. Dies betrifft sowohl die Herleitung deduktiver Kategorien aus Theorien als auch die induktive Ableitung aus dem Material. Eine detaillierte Dokumentation ermöglicht es, den Analyseprozess nachzuvollziehen und die Ergebnisse intersubjektiv zu überprüfen (Mayring 2015b).

Weiterhin ist die *Kommunikative Validierung* von Bedeutung. Ergebnisse der Analyse können mit den untersuchten Personen oder mit Expertinnen und Experten diskutiert werden, um zu prüfen, ob die Interpretationen überzeugend und angemessen erscheinen (Kuckartz 2018). Auch die *Triangulation* durch die Kombination unterschiedlicher Datenquellen, Methoden oder Forscherinnen und Forscher kann zur Stärkung der Validität beitragen.

Hinsichtlich der *Reliabilität* wird insbesondere auf die Konsistenz der Kategorienanwendung geachtet. Dies geschieht durch die Entwicklung präziser Kodierregeln, eine Einweisung bzw. Schulung der Kodierenden sowie durch den Einsatz von Intercoder-Übereinstimmungen. Solche Verfahren erhöhen die Nachvollziehbarkeit und verringern die Gefahr subjektiver Verzerrungen (Schreier 2012).

Schließlich spielt die *Reflexivität* der Forschenden eine wichtige Rolle. Da die qualitative Inhaltsanalyse nicht ohne interpretative Entscheidungen auskommt, ist es notwendig, dass sich Forschende ihrer eigenen Vorannahmen bewusst sind und diese im Analyseprozess offenlegen.

Die qualitative Inhaltsanalyse nach Mayring wird in dieser Arbeit in zwei zentralen Untersuchungskontexten eingesetzt: zur Analyse der Entscheidungsgrundlagen und Charakteristika von Feedback durch Lehrpersonen (Kapitel 4) sowie zur Analyse von Rückmeldungen, die durch große Sprachmodelle (LLMs) generiert werden (Kapitel 6). In beiden Fällen bietet die Methode eine geeignete Grundlage, um komplexe textbasierte Daten systematisch zu erschließen und sowohl theoriegeleitet als auch materialoffen auszuwerten:

Die Kombination aus deduktiver und induktiver Kategorienbildung dient in dieser Arbeit sowohl der materialsensiblen Analyse, als auch dem Ziel, ein belastbares Kategoriensystem zu entwickeln, an das künftige Studien anschließen können. Durch die Anlehnung an etablierte Modelle wird Anschlussfähigkeit an den bestehenden Forschungsstand gewährleistet, während die induktiven Erweiterungen sicherstellen, dass auch kontextspezifische Besonderheiten von Lehrpersonen- und LLM-Rückmeldungen erfasst werden. Auf diese Weise entsteht eine methodisch abgesicherte Grundlage, die nicht nur für die vorliegende Untersuchung tragfähig ist, sondern zugleich Vergleichbarkeit und Weiterentwicklung in zukünftiger Forschung ermöglicht.

3.3.3 Konzeptuelle Modellierung

Die konzeptuelle Modellierung ist ein methodisches Vorgehen, das darauf abzielt, komplexe Gegenstandsbereiche in formalisierten, maschinenlesbaren Strukturen abzubilden. Ihr Ziel besteht darin, relevante Entitäten, Relationen und Regeln so zu repräsentieren, dass sie einerseits den inhaltlichen Kern eines Phänomens erfassen und andererseits für algorithmische Verarbeitung zugänglich gemacht werden können (Pastor 2008). In der vorliegenden Arbeit wird die Methodik eingesetzt, um (Lern-)Aufgaben, Antworten sowie instruktionale und individuelle Faktoren beim Programmierlernen systematisch zu beschreiben und damit eine Grundlage für kontextsensitive Feedbackprozesse zu schaffen.

Ursprünglich in der Informatik und Wirtschaftsinformatik verankert, entwickelte sich die konzeptuelle Modellierung aus dem Bedürfnis, reale Anwendungsdomänen in strukturierte, formale Modelle zu überführen (Wand und Weber 1993; Lindland, Sindre und Solvberg 1994). Dabei wird nicht nur eine Abbildung der Realität angestrebt, sondern auch eine Abstraktion, die für bestimmte Fragestellungen besonders relevant ist. Konzepte wie Metamodellierung, Ontologien oder semantische Netze haben die Methode in den letzten Jahrzehnten geprägt und auch in bildungswissenschaftlichen Kontexten an Bedeutung gewonnen (Devedzic, Djuric und Gasevic 2009; Uschold und Grüninger 1996).

3.3.3.1 Charakteristische Merkmale

Die konzeptuelle Modellierung ist durch mehrere zentrale Eigenschaften gekennzeichnet:

- **Abstraktion:** Reduktion komplexer Realitäten auf für die Fragestellung relevante Elemente.
- **Strukturierung:** Systematische Anordnung von Entitäten, Relationen und Regeln.
- **Formalisierung:** Präzise, wohldefinierte Beschreibung, die maschinell interpretierbar ist.
- **Transparenz:** Dokumentierte Definitionen und Repräsentationen, die Nachvollziehbarkeit ermöglichen.
- **Anschlussfähigkeit:** Verwendung standardisierter Repräsentationsformen (z. B. UML, Ontologiesprachen, logische Formalismen), die Interoperabilität sichern.

3.3.3.2 Vorgehen und Techniken

Typischerweise umfasst die konzeptuelle Modellierung mehrere Schritte, die von der Erhebung domänenspezifischer Begriffe bis hin zur formalen Überprüfung reichen. Zunächst steht die *Domänenanalyse*, in der die relevanten Objekte, Prozesse und Konzepte identifiziert werden. Darauf folgt die *Abstraktion und Strukturierung*, bei der zentrale Entitäten, Relationen und Kardinalitäten herausgearbeitet und in einer konsistenten Form angeordnet werden. Im nächsten Schritt erfolgt die *Formalisierung*, d. h. die präzise und wohldefinierte Spezifikation der Konzepte in einer maschinenlesbaren Repräsentation. Dieser Schritt gilt in der Literatur als konstitutiv: Nur durch eine formale, computerprozessierbare Darstellung lassen sich Modellierungsmethoden plattformunabhängig implementieren, vergleichen und algorithmisch weiterverarbeiten (Döller, Karagiannis und Utz 2023). Abschließend wird das Modell einer *Validierung* unterzogen, bei der Konsistenz, Vollständigkeit und Übereinstimmung mit der intendierten Domäne überprüft werden. Auf diese Weise verbindet konzeptuelle Modellierung inhaltliche Präzision mit technischer Anschlussfähigkeit.

3.3.3.3 Gütekriterien

Die Qualität konzeptueller Modelle lässt sich nach dem Rahmenmodell von Lindland, Sindre und Solvberg (1994) in drei grundlegende Dimensionen unterscheiden: syntaktische, semantische und pragmatische Qualität.

Die *syntaktische Qualität* beschreibt, in welchem Maße ein Modell den Regeln der verwendeten Modellierungssprache folgt. Ziel ist die syntaktische Korrektheit, das heißt, dass alle im Modell enthaltenen Aussagen wohlgeformt sind und den Grammatikregeln der Sprache entsprechen. Die *semantische Qualität* bezieht sich auf das Verhältnis zwischen Modell und Domäne. Ein Modell ist dann von hoher semantischer Qualität, wenn seine Aussagen korrekt und relevant in Bezug auf das zu modellierende Problem sind (*Validität*) und wenn es alle relevanten und richtigen Aussagen der Domäne enthält (*Vollständigkeit*). Da eine vollständige Erfassung in der Praxis kaum erreichbar ist, führen die Autoren den Begriff der *feasible validity* und *feasible completeness* ein, die eine realistische Abwägung von Nutzen und Aufwand erlauben. Die *pragmatische Qualität* schließlich betrifft die Verständlichkeit des Modells für seine Adressaten. Wesentliches Ziel ist die *Comprehension*, also dass alle relevanten Akteurinnen und Akteure das Modell tatsächlich verstehen. Selbst ein semantisch korrektes Modell wäre ohne hinreichende Verständlichkeit nicht brauchbar. Darüber hinaus betonen die Autoren die Bedeutung der *formalen Prüfbarkeit*. So können etwa syntaktische Korrektheit und bestimmte Konsistenzprüfungen automatisiert erfolgen, wenn das Modell in einer formal definierten Sprache vorliegt. Dadurch lassen sich Fehler systematisch erkennen und die Qualität des Modells methodisch absichern.

Im Rahmen dieser Arbeit wird konzeptuelle Modellierung eingesetzt, um die Vielzahl an Einflussfaktoren von Feedback beim Programmierlernen in formalisierten Strukturen abzubilden und damit für regel- und datenbasierte Verfahren zugänglich zu machen. Dabei geht es nicht nur um die Erfassung der Aufgaben und Antworten selbst, sondern ebenso um die instruktionalen und individuellen Bedingungen, die maßgeblich bestimmen, welche Feedback-Strategien angemessen und wirksam sind (Narciss 2018). Die konzeptuelle Modellierung stellt somit das methodische Bindeglied dar, um diese Faktoren in ein strukturiertes, maschinenlesbares Modell zu überführen, das als Grundlage für die Entwicklung, Auswahl und Überprüfung unterschiedlicher Feedback-Strategien dient. Auf diese Weise soll ermöglicht werden, die Komplexität von Feedbackprozessen systematisch zu erfassen und zugleich für automatisierte, regelgeleitete Entscheidungen nutzbar zu machen.

Kapitel 4

Entscheidungsgrundlagen und Charakteristika von Feedback durch Lehrpersonen

Das ITFL-Modell beschreibt Feedback als Bestandteil eines externen Regelkreises, in dem auf Basis beobachtbarer Lernhandlungen Abweichungen zwischen Ist- und Soll-Zuständen diagnostiziert und durch geeignete Rückmeldungen adressiert werden. Ein zentraler Schritt in diesem Regelkreis besteht in der Entscheidung, *ob* eingegriffen wird und – falls ja – *welche* Form von Feedback im jeweiligen Kontext lernwirksam ist. Damit eine solche Entscheidung empirisch fundiert modelliert werden kann, ist zu klären, welche Indikatoren ihr zugrunde liegen. Im Kontext der Programmierausbildung ist bislang jedoch nur wenig untersucht, auf welcher Grundlage Lehrpersonen solche Entscheidungen tatsächlich treffen. Es fehlt systematisches Wissen darüber, in welchen Situationen Feedback gegeben wird, welche Aspekte des Codes oder des Bearbeitungsprozesses dabei berücksichtigt werden und wann bzw. warum bewusst auf eine Intervention verzichtet wird. Für die Weiterentwicklung automatisierten, kontextsensitiven Feedbacks und adaptiver Systeme ist ein vertieftes Verständnis dieser Entscheidungsprozesse daher zentral.

Die vorliegende Untersuchung verfolgt das Ziel, eine empirische Grundlage zur Modellierung adaptiver Feedback-Strategien für das Programmierenlernen zu schaffen. Im Zentrum steht die Frage, auf welcher Basis erfahrene Lehrpersonen entscheiden, ob sie Lernenden Feedback geben, und wie sie dieses inhaltlich ausgestalten. Daraus ergibt sich das Ziel, ein empirisch fundiertes Kategoriensystem menschlicher Feedbackpraktiken zu entwickeln. Dieses Kategoriensystem soll die Vielfalt pädagogischer Rückmeldungen systematisch erfassen und damit eine Grundlage für die didaktisch fundierte Modellierung adaptiver Lernumgebungen bilden – etwa im Rahmen automatisierter Tutorsysteme oder LLM-basierter Lernassistenz. Der Fokus liegt dabei auf der deskriptiven Erfassung realer Feedbackpraktiken, nicht auf der normativen Bestimmung „richtiger" oder „falscher" Strategien.

Die Studie knüpft damit direkt an die in Abschnitt 3.2 formulierte Forschungsfrage **FF1** an, die sich auf die Entscheidungsgrundlagen und Charakteristika von Feedback durch Lehrpersonen richtet. Die nachfolgend beschriebenen methodischen Schritte (siehe Abschnitt 4.2) sind entsprechend darauf ausgerichtet, empirische Antworten auf diese Fragestellung zu gewinnen. Die Untersuchung ist dabei im *Relevance Cycle* des Design-Science Research (vgl. Abschnitt 3.3.1) zu verorten: Sie nimmt ein praxisrelevantes Problemfeld auf – die Entscheidungsprozesse und

Teile dieses Kapitels wurden bereits vorab veröffentlicht in Lohr et al. (2024a) und Lohr et al. (2025a). Für Details siehe Übersicht zu Vorabveröffentlichungen auf Seite VII.

Feedbackpraktiken von Lehrpersonen – und untersucht dieses systematisch. Zugleich trägt sie durch die Entwicklung eines Kategoriensystems zur Wissensbasis im *Rigor Cycle* bei, indem sie empirische Evidenz in strukturierte, theoriegeleitete Kategorien überführt. Damit bildet die Untersuchung die empirische Grundlage für die anschließende konzeptuelle Modellierung (Kapitel 5) sowie für die technologische Erprobung kontextsensitiver Feedback-Strategien (Kapitel 6).

4.1 Stand der Forschung

Wie in Abschnitt 2.1 ausgeführt, gilt die Reduktion der Diskrepanz zwischen einem aktuellen Lern- oder Leistungsstand (IST) und einem angestrebten Zielzustand (SOLL) als zentrales Wirkprinzip von Feedback (Ramaprasad 1983; Sadler 1989; Hattie und Timperley 2007). Aufbauend auf dieser Grundannahme zeigen Meta-Analysen und Reviews, dass Feedback insbesondere dann lernwirksam ist, wenn es zielgerichtet, spezifisch, aufgaben- oder prozessnah, zeitnah, nicht-wertend und handlungsleitend gegeben wird (Kluger und DeNisi 1996; Shute 2008; Nicol und Macfarlane-Dick 2006; Evans 2013; Boud und Molloy 2013). Weitere Faktoren wie die Glaubwürdigkeit der Quelle oder die curriculare Einbettung gelten ebenfalls als zentrale Voraussetzungen für eine wirksame Nutzung von Feedback (Black und Wiliam 1998; Ilgen, Fisher und Taylor 1979).

Diese Befunde liefern belastbare, aber zugleich sehr generische Prinzipien. Sie lassen offen, welche konkreten Situationen im Lernprozess tatsächlich Rückmeldeanlässe darstellen, nach welchen Heuristiken Lehrpersonen in der Praxis Entscheidungen treffen und auf welchem Granularitätsniveau Rückmeldungen angemessen sind. Für die Entwicklung domänenspezifischer Feedback-Strategien genügt es daher nicht, allein auf allgemeine Leitlinien zurückzugreifen; vielmehr bedarf es einer empirisch fundierten Analyse von Entscheidungssituationen in konkreten Praxisfeldern.

Über die generischen Leitlinien hinaus gibt es erste empirische Untersuchungen, die das Handeln von Expertinnen und Experten in realen Feedbacksituationen analysieren. D'Mello, Lehman und Person (2010) untersuchten hierzu die Strategien von zehn erfahrenen Tutorinnen und Tutoren in insgesamt fünfzig Sitzungen verschiedener Fächer (u. a. Algebra, Geometrie, Physik, Chemie und Biologie). Im Fokus stand die Frage, ob Feedback direkt und unmittelbar oder indirekt und verzögert gegeben wurde und ob sich Unterschiede zwischen den Domänen zeigen. Die Ergebnisse verdeutlichen, dass das Feedback durchgehend *direkt*, *unmittelbar*, *differenziert* und weitgehend *bereichsunabhängig* erfolgt. Dies weist darauf hin, dass Lehrpersonen in situativen Entscheidungsmomenten konsistente Muster anwenden, die weniger aus inhaltlichen Besonderheiten der Domäne als vielmehr aus allgemeinen didaktischen Orientierungen hervorgehen.

Für den Bereich der Programmierausbildung ist jedoch bislang wenig darüber bekannt, wie solche Muster konkretisiert werden. Erste Arbeiten adressieren diese Lücke im Kontext blockbasierter Programmierumgebungen: Dong et al. (2021) rekonstruierten auf Basis von Logdaten und Expertenratings die Gründe, aus denen Lehrpersonen während der Bearbeitung intervenieren. Sie identifizierten sechs zentrale Entscheidungskategorien: (1) das Fehlen wesentlicher Komponenten (*missing key components*), (2) die Verwendung unnötiger Blöcke (*using unnecessary components*), (3) die fehlerhafte Nutzung benötigter Blöcke (*misusing needed components*), (4) das Vorliegen von Logikfehlern (*logical error*), (5) eine unklare Intention der Lernenden (*unknown intent*) sowie (6) der Bedarf an Hinweisen oder Vorschlägen für nächste Schritte (*hint needed*). Diese Kategorien verdeutlichen, dass Feedbackentscheidungen nicht allein aus allgemeinen Prinzipien ableitbar sind, sondern maßgeblich von situativen Merkmalen der Bearbeitung und wahrgenommenen Lernbarrieren abhängen können. Zugleich zeigt sich, dass Lehrpersonen ihre Interventionen nicht nur reaktiv am Endprodukt (z. B. Testergebnisse), sondern proaktiv am Prozess der Aufgabenbearbeitung ausrichten. Darüber hinaus berichten Dong et al. Evidenz für die Bedeutung von

positivem Feedback: Lernende mit korrekten, aber unvollständigen Programmen neigten dazu, ihre Arbeit zurückzusetzen, um später erneut denselben Zwischenstand zu erreichen. Als zusätzlichen Interventionsgrund schlagen sie daher (7) die *Bestätigung, dass ein Teilziel erreicht wurde* (positive Bestätigung partieller Korrektheit) vor, ergänzt um Eingriffe bei destruktivem Bearbeitungsverhalten.

Neben Untersuchungen in blockbasierten Umgebungen finden sich auch Studien im Kontext textueller Programmierung. Jeuring et al. (2022) zielten darauf, Leitlinien für den *Zeitpunkt* und die *Art* von Interventionen während textbasierter Programmieraufgaben zu entwickeln. Dazu annotierten erfahrene Lehrpersonen Datensätze mit Sequenzen von Programmier-Schritten (*steps*) und markierten, an welchen Stellen sie eingreifen würden und in welcher Form (z. B. Fehlerkorrektur, Hinweis, Bestätigung). Auf diese Weise entstand ein Korpus von Interventionspunkten, das als Grundlage für Richtlinien diente. Die Ergebnisse zeigten jedoch deutliche Uneinigkeiten zwischen den Beteiligten, etwa in der Frage, ob unmittelbar nach fehlerhaften Schritten eingegriffen werden sollte oder ob den Lernenden zunächst produktiver „Struggle" ermöglicht werden sollte, in welchem Umfang positives Feedback auf Subziel-Ebene erforderlich sei oder wie mit wiederholten Fehlern umzugehen sei.

Neben der Frage, *wann* und *warum* Feedback gegeben wird, stellt sich auch die Frage, *wie* Feedback inhaltlich ausgestaltet ist. Im Bereich der Programmierausbildung liegen hierzu bislang überwiegend Untersuchungen zu Online-Lernplattformen und automatisierten Systemen vor. Wie in Abschnitt 2.3 dargestellt, beschränkt sich das in Lernplattformen implementierte Feedback meist auf *Simple Feedback* (Keuning, Jeuring und Heeren 2019) (z. B. Korrektheitsrückmeldungen, Testergebnisse oder Musterlösungen) und bleibt damit in seiner didaktischen Reichweite begrenzt. Auch die Untersuchung von Jeuring et al. (2022) verdeutlicht, dass automatisierte Systeme primär einfache, regelbasierte Rückmeldungen umsetzen, während komplexere, prozessorientierte Strategien schwer operationalisierbar bleiben.

Darüber hinaus zeigt Kiesler (2021), dass in bestehenden Lernsystemen insbesondere *strategisches Feedback* weitgehend fehlt. Gemeint sind hier sowohl Rückmeldungen, die auf die *Entwicklung und Anwendung effektiver Lösungsstrategien* abzielen, als auch solche, die selbst *unter Berücksichtigung der jeweils erkennbaren Lernstrategien* formuliert werden. Beide Dimensionen – Feedback *zu* Strategien und Feedback *unter* Berücksichtigung von Strategien – bilden zentrale, bislang kaum erschlossene Elemente einer kontextsensitiven Rückmeldepraxis. Kiesler (2021) adressiert diesen Mangel durch die Einführung der Kategorie *Strategic Processing Steps*, die gezielt auf die Förderung metakognitiver und strategischer Prozesse im Programmierlernen ausgerichtet ist.

Während für automatisiertes Feedback in digitalen Lernumgebungen mittlerweile eine Reihe von Klassifikationen vorliegen (vgl. Abschnitt 2.1.2.2), fehlt bislang ein entsprechendes, empirisch fundiertes System zur Beschreibung *menschlicher* Feedbackpraktiken in realen Programmierprozessen. Allgemeine Taxonomien von Feedbacktypen erfassen zwar die Bandbreite möglicher Rückmeldeformen, lassen jedoch offen, ob und wie diese tatsächlich in der Praxis erfahrener Lehrpersonen auftreten. Unklar bleibt insbesondere, in welchem Verhältnis unterschiedliche Formen – etwa korrektives, hinweisendes, erklärendes oder bestätigendes Feedback – zueinander stehen und wie sie in situativen Entscheidungskontexten miteinander kombiniert werden.

Damit stellt sich die Frage, ob bestehende Klassifikationen den spezifischen Anforderungen des Programmierenlernens gerecht werden oder ob sie durch ein empirisch fundiertes Kategoriensystem erweitert werden müssen, das die Entscheidungslogik und Rückmeldepraxis erfahrener Lehrpersonen abbildet. Die bisherigen Befunde legen nahe, dass die Identifikation typischer Interventionssituationen und Entscheidungsmuster im Programmierprozess eine zentrale Voraussetzung darstellt, um die Frage nach dem *richtigen Zeitpunkt*, den *Gründen* und den *Inhalten* von Feedback systematisch untersuchen und beantworten zu können.

4.2 Methodik

Die Untersuchung folgt einer qualitativ-explorativen Forschungslogik, die bestehende theoretische Kategorien als heuristischen Bezugsrahmen nutzt und diese empirisch überprüft sowie erweitert. Ausgangspunkt bilden deduktive Kategorien aus der Feedbackforschung, die im Verlauf der Analyse durch induktiv entwickelte Ergänzungen präzisiert und ausdifferenziert werden. Dieses kombinierte Vorgehen entspricht der in der qualitativen Sozialforschung etablierten Logik einer deduktiv-induktiven Kategorienbildung (siehe Abschnitt 3.3.2).

Zentral ist dabei die Arbeit mit realitätsnahen Problemlösungsverläufen von Programmieranfängerinnen und -anfängern. Unter *realitätsnah* sind dabei Szenarien zu verstehen, die typische Herausforderungen und Bearbeitungsweisen aus dem schulischen und hochschulischen Programmierunterricht abbilden. Es sollen somit keine künstlich konstruierten Modellfälle verwendet werden, sondern authentische Bearbeitungsverläufe, die den Handlungskontext von Lehrpersonen angemessen widerspiegeln können. Das Studiendesign ermöglicht dadurch eine Analyse pädagogischer Urteilsbildung im Kontext typischer Lernsituationen beim Programmieren.

Zur Beantwortung der Forschungsfragen **FF1.1** und **FF1.2** (siehe Abschnitt 3.2) wird eine qualitative Online-Erhebung mit erfahrenen Lehrpersonen durchgeführt. Grundlage der Erhebung sind dabei Bearbeitungsverläufe von Programmieranfängerinnen und -anfängern, die in Form von Schritt-für-Schritt-Protokollen (sog. Codezustandsfolgen) präsentiert werden. Für jeden dargestellten Codezustand werden die Lehrpersonen aufgefordert, die folgenden drei Fragen zu beantworten:

1. Würden Sie an dieser Stelle Feedback geben? (Ja / Nein / Vielleicht)
2. Warum würden Sie (k)ein Feedback geben? (Begründung – „Why?“)
3. Wie würde dieses Feedback aussehen? (Formulierung – „How?“)

Die Auswertung der Antworten erfolgt mittels qualitativer Inhaltsanalyse nach Mayring (2015b) (vgl. Abschnitt 3.3.2). Die deduktiv abgeleiteten Kategorien bilden dabei den Ausgangspunkt für die Annotation. Zugleich werden neue Aspekte induktiv ergänzt. Die resultierenden Kategoriensysteme dienen nicht nur der strukturierten Beschreibung der Rückmeldungen, sondern stellen eine konzeptuelle Grundlage für die in Kapitel 5 vorgestellte Modellierung von Programmieraufgaben, Antworten und Feedback-Strategien dar.

4.2.1 Datenbasis und Datenauswahl

Grundlage der Untersuchung bilden Verläufe protokollierter Programmieraktivitäten aus realen Lernkontexten von Programmieranfängerinnen und -anfängern. Um realitätsnahe Feedbackreaktionen zu ermöglichen, war es das Ziel, den Lehrpersonen nachvollziehbare und didaktisch relevante *Ausschnitte* aus diesen Problemlösungsprozessen zu präsentieren, wie sie in aufgezeichneten Programmierverläufen (z. B. in Form von Snapshots oder Diff-Darstellungen) sichtbar werden. Mit *didaktisch relevant* sind Ausschnitte aus den Programmierverläufen gemeint, die typischerweise bei Erlernen der Programmierung auftreten und aus lernpsychologischer oder instruktionaler Sicht eine potenzielle Gelegenheit für Feedback oder Lernunterstützung bieten. Dazu zählen beispielsweise das Auftreten bekannter Fehlermuster, wiederholte Versuche zur Problemlösung, stagnierende Bearbeitungsphasen oder suboptimale Lösungsstrategien (Ettles, Luxton-Reilly und Denny 2018; Neuwinger und Riehle 2025; McCall und Kölling 2019). Solche Situationen sind nicht nur charakteristisch für die Zielgruppe, sondern auch anschlussfähig für eine didaktische Intervention, da sie potenziell lernrelevante Hürden oder Missverständnisse sichtbar machen.

Da einzelne Bearbeitungsverläufe oftmals mehrere Hundert Codezustände umfassen, war eine Reduktion auf eine annotierbare Anzahl von Code-Ausschnitten (im Folgenden als *Steps* bezeichnet)

erforderlich. Die Auswahl und Aufbereitung der Daten für die Umfrage folgte einer systematischen, mehrstufigen Vorgehensweise, deren einzelne Schritte in den folgenden Abschnitten näher erläutert werden:

- Auswahl geeigneter Datensätze und -sequenzen (Abschnitt 4.2.1.1 sowie 4.2.1.2)
- Aufbereitung der Sequenzen auf Keystroke-Ebene (Abschnitt 4.2.1.3)
- Reduktion der Sequenzen auf relevante *Steps* (Abschnitt 4.2.1.4)
- Nachbildung der *Steps* in den ursprünglichen Lernumgebungen (Abschnitt 4.2.1.5)

Die detaillierte Beschreibung und Begründung dieses Vorgehens folgt den Anforderungen an methodische Transparenz und Nachvollziehbarkeit, wie sie in der qualitativen Inhaltsanalyse nach Mayring formuliert sind (vgl. Abschnitt 3.3.2). Dadurch wird gewährleistet, dass die Auswahl und Aufbereitung der Daten nicht nur pragmatisch motiviert, sondern auch wissenschaftlich methodisch abgesichert ist.

4.2.1.1 Auswahl der Datensätze

Für die Untersuchung werden Datensätze herangezogen, die spezifischen Anforderungen genügen müssen, um valide und aussagekräftige Ergebnisse zur Analyse von Feedbackentscheidungen zu ermöglichen. Im Sinne des Relevance Cycle (Abschnitt 3.3.1) werden dabei solche Datensätze priorisiert, die praxisnahe und zugleich wissenschaftlich verwertbare Evidenz liefern. Die folgenden Kriterien leiten die Auswahl:

1. enthält Bearbeitungsprotokolle von **textueller Programmierung**
   Im Vergleich zu blockbasierten Umgebungen erfordert textuelle Programmierung ein höheres Maß an syntaktischem Verständnis, wodurch differenziertere Einsichten in typische Fehler und Lernschwierigkeiten in frühen Phasen des Programmierenlernens ermöglicht werden.
2. enthält **reale** Bearbeitungsprotokolle von Programmieranfängerinnen und -anfängern
   Realitätsnahe Datensätze erhöhen die externe Validität der Untersuchung, da sie tatsächliche Interaktionen mit Lernumgebungen abbilden statt künstlich erzeugter Beispiele.
3. jede Sequenz enthält **aufeinanderfolgende *Steps*** beim Lösen von Programmieraufgaben
   Durch die Verfügbarkeit aufeinanderfolgender Bearbeitungsschritte wird es möglich, auch den Verlauf der Problemlösung zu analysieren und so Feedbackentscheidungen im jeweiligen situativen Kontext besser nachzuvollziehen – etwa bei typischen Fehlersequenzen oder Wiederholungen.
4. ist **öffentlich verfügbar**
   Die öffentliche Verfügbarkeit der Datengrundlage stellt sicher, dass die entwickelten Kategoriensysteme und Analyseansätze transparent geprüft, diskutiert und weiterentwickelt werden können.
5. enthält **Aufgaben für Programmierlernende** in verschiedenen Schwierigkeitsgraden, welche im Rahmen der ersten Wochen beim Programmierenlernen zum Einsatz kommen und zentrale Basiskonzepte enthalten
   Aufgaben aus der frühen Lernphase fokussieren grundlegende Konzepte (wie z. B. Kontrollstrukturen, Rekursion), bei denen typischerweise ein hoher Unterstützungsbedarf besteht (Lahtinen, Ala-Mutka und Järvinen 2005) und bieten daher ein besonders reichhaltiges Spektrum möglicher Feedbacksituationen.
6. stellt idealerweise **Kontextinformationen** bereit, wie Informationen über die Lernenden und über die zugehörige Lernplattform
   Diese Informationen ermöglichen die Rekonstruktion der Bearbeitungsschritte innerhalb der ursprünglichen Lernplattform sowie die Einbindung eines Lernendenprofils, um die Situation für erfahrene Lehrpersonen möglichst realitätsnah darzustellen. Auf diese Weise lassen sich Feedbackentscheidungen besser im Kontext individueller Vorerfahrungen, früherer Interaktionen und verfügbarer Hilfestellungen interpretieren.

7. deckt **verschiedene Programmiersprachen** ab
   Eröffnet die Möglichkeit, auch sprachspezifische Fehlertypen (z. B. Einrückungsfehler in Python) und damit verbundene Feedback-Strategien zu berücksichtigen. Gleichzeitig erlaubt sie die Untersuchung, inwieweit sich bestimmte Rückmeldungen über sprachliche Kontexte hinweg generalisieren lassen.

Eine Sichtung mehrerer öffentlich verfügbarer Datensätze führte zur Auswahl der in Tabelle 4.1 aufgeführten Datengrundlagen, die in den folgenden Abschnitten näher beschrieben werden. Die Auswahl orientierte sich an den zuvor definierten Kriterien, wurde jedoch auch durch pragmatische Erwägungen bestimmt: Zwei der Datensätze lagen bereits in aufbereiteter Form vor, während andere vergleichbare Datensätze (mit ähnlichen Fehlerbildern) lediglich als Rohdaten auf Keystroke-Ebene verfügbar waren.

ID	Quelle	Lizenz	Sprache	Granularität
`CodingBat`	(Kiesler 2022), (Kiesler, Goethe-Universität Frankfurt Am Main und Hochschule Fulda 2022)	©DZHW[2]	`Java`	Token$^+$
`FITech`	FITech 101 Introduction to Programming [1]	CC BY-SA 4.0	`Dart`	Keystroke
`TaskTracker`	(Lyulina et al. 2021)	MIT	`Python`	Keystroke

[1]https://fitech101.aalto.fi/
[2]https://www.dzhw.eu/

Tabelle 4.1: Ausgewählte Datensätze für die Erhebung

Der `CodingBat`-Datensatz enthält Daten von Programmieranfängerinnen und -anfängern, die `Java`-Aufgaben im Rahmen einer kontrollierten Think-Aloud-Studie bearbeiteten (Kiesler, Goethe-Universität Frankfurt Am Main und Hochschule Fulda 2022; Kiesler 2022). CodingBat ist eine Programmierlernumgebung, die von Parlante[1] entwickelt wurde. Dabei handelt es sich um Übungsaufgaben aus einer einführenden Programmierlehrveranstaltung der Stanford Universität. Die Lernumgebung bietet einfaches Feedback über den Erfolg bzw. Misserfolg von Unit-Tests. Der gesamte Datensatz umfasst zwei Aufgaben (`Fibonacci` und `Fakultät von n`) mit insgesamt 16 Bearbeitungssequenzen auf Token-Level-Granularität, die auf Basis von Videoaufzeichnungen erstellt wurden. Zusätzlich zu den Bearbeitungsschritten wurden auch Aktionen der Lernenden wie *compile*, *get hint* und *show solution* protokolliert. Der Datensatz ist ausschließlich für die wissenschaftliche Nutzung bestimmt und im Rahmen einer Registrierung beim DZHW frei zugänglich [2] – die Lizenz erlaubt jedoch keine vollständige Veröffentlichung von annotierten Versionen.

Der `FITech`-Datensatz wurde im Rahmen eines Online-Programmierkurses der Aalto-Universität erhoben. Der Kurs behandelt die Grundlagen der Programmierung mit der Programmiersprache `Dart` und umfasst 64 verschiedene kleinere Programmieraufgaben. Diese decken grundlegende Konzepte des Einstiegs in die Programmierung ab, darunter elementare Ein- und Ausgabeverfahren, Kontrollstrukturen wie Bedingungen und Wiederholungen, die Definition und Nutzung von Methoden sowie den Umgang mit grundlegenden Datenstrukturen. Die Daten wurden von einer in eine Kursplattform für Fernunterricht eingebetteten Online-IDE erfasst. Erhoben wurden sowohl Codebearbeitungen auf Keystroke-Ebene als auch Interaktionen der Lernenden mit der Lernplattform – etwa *run*-, *submit*- oder *request help*-Aktionen.

[1]`https://codingbat.com/about.html`
[2]Sie können unter Berücksichtigung der Richtlinien für die Datennutzung über die Website des FDZ beantragt werden. Für die Nutzung von SUF wird ein Datennutzungsvertrag abgeschlossen.

Der `TaskTracker`-Datensatz umfasst die Bearbeitungsschritte von 148 Studierenden in insgesamt sechs Programmieraufgaben, die in den Programmiersprachen `Python`, `Java`, `Kotlin` und `C++` vorliegen (Lyulina et al. 2021). Die Aufzeichnung erfolgte über ein Plugin für IntelliJ-basierte IDEs auf Keystroke-Level. In der Aufgabenbeschreibung standen den Studierenden drei Testfälle zur Verfügung, die sie durch Codeausführung, Eingabe und manuelle Überprüfung der Ausgabe kontrollieren konnten.

4.2.1.2 Auswahl der Sequenzen

Im nächsten Schritt wurden aus den ausgewählten Datensätzen konkrete *Sequenzen* bestimmt, die für die Analyse von Feedbackentscheidungen geeignet erscheinen. Ziel war es, Bearbeitungssituationen zu identifizieren, die für Lehrpersonen nachvollziehbar sind und potenzielle Anlässe für Rückmeldungen enthalten. Darüber hinaus sollte eine Variation an Lösungsansätzen abgebildet werden, um unterschiedliche Bearbeitungsstile und Strategien der Lernenden vergleichen und adressieren zu können.

Im Folgenden werden die zentralen Auswahlkriterien aufgeführt, die bei der Zusammenstellung der Sequenzen berücksichtigt werden, jeweils ergänzt um eine kurze Begründung ihrer Relevanz für die Analyse von Feedbacksituationen.

- Die zugehörige Aufgabe ist **für die ersten Wochen einer Programmierlehrveranstaltung konzipiert**.
  Gemeint sind Aufgaben, die keine vertieften Vorkenntnisse voraussetzen und grundlegende Programmierkonzepte fokussieren. Solche Aufgaben bilden häufig typische Herausforderungen der frühen Lernphase ab, in denen Feedback eine besondere Relevanz haben kann (Hattie und Timperley 2007).

- Die **Musterlösung** der Aufgabe umfasst zwischen **10 und 15 Codezeilen**.
  Eine solche Länge bietet in der Regel genügend Raum für unterschiedliche Lösungsstrategien und typische Fehlerbilder, bleibt jedoch überschaubar genug, damit erfahrene Lehrpersonen die Bearbeitung nachvollziehen können.

- Die Sequenz enthält Situationen, in denen Lernende **deutlich erkennbare Schwierigkeiten** aufweisen oder Feedback anfordern.
  Solche Passagen stellen potenzielle Gelegenheiten für Rückmeldungen dar – etwa zur Unterstützung, Korrektur oder Orientierung – und sind damit besonders interessant für die Analyse von Feedbacksituationen.

- Ein **erfolgreicher Abschluss** der Aufgabe ist **nicht erforderlich**.
  Auch unvollständige oder abgebrochene Lösungsversuche können Hinweise auf typische Fehlerverläufe und mögliche Anlässe für Feedback liefern.

- Sequenzen, bei denen der **Lösungsprozess von Beginn an** dokumentiert ist, werden bevorzugt.
  Der Einstieg in eine Aufgabe ist häufig mit Unsicherheiten verbunden und bietet damit relevante Ansatzpunkte für Feedback. Zudem dürfte die vollständige Sichtbarkeit des Bearbeitungsprozesses den Teilnehmenden das Nachvollziehen der Lösungswege im jeweiligen Kontext erleichtern.

Eine Sichtung aller verfügbaren Sequenzen in den ausgewählten Datensätzen anhand der genannten Kriterien führte zur Auswahl von insgesamt sechs Bearbeitungsverläufen – jeweils zwei Sequenzen zu drei unterschiedlichen Aufgaben. Die Auswahl von zwei Sequenzen pro Aufgabe ermöglicht es, unterschiedliche Bearbeitungswege innerhalb derselben Aufgabenstellung zu vergleichen. So lassen sich Unterschiede in Lösungsstrategien, Fehlerbildern und potenziellen Feedbackanlässen herausarbeiten, ohne dass der inhaltliche Kontext variiert. Zugleich eröffnet die Aufgabenvielfalt den Vergleich zwischen verschiedenen Programmiersprachen (`Java`, `Dart`, `Python`) und konzeptuellen Schwerpunkten. Tabelle 4.2 bietet einen Überblick über die ausge-

Aufgabe	Sprache	Beschreibung	Datensatz	Konzepte	Sequenzen
Fibonacci	`Java`	Berechnung der n-ten Fibonacci-Zahl mit Rekursion.	`CodingBat`	Methoden, Bedingungen, Rekursion	B02, B05
Password	`Dart`	Schreiben einer Methode, die so lange nach einem Passwort fragt, bis dieses korrekt eingegeben wird.	`FITech`	Methoden, Bedingungen, Parameter, Wiederholung, input/output	148, 538
MaxDigit	`Python`	Größte Ziffer innerhalb eines Strings aus Zahlen identifizieren und ausgeben.	`TaskTracker`	Bedingungen, Wiederholung, Input/Output	tt1, tt2

Tabelle 4.2: Übersicht der ausgewählten Aufgaben und Sequenzen

Fibonacci

```
The fibonacci function is a famous bit of mathematics, and it happens to
have a recursive definition. The first two values in the sequence are 0 and
1 (essentially two base cases). Each subsequent value is the sum of the two
previous values, so the whole sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21 and
so on. Define a recursive fibonacci(n) method that returns the nth fibonacci
number, with n=0 representing the start of the sequence.
fibonacci(0)  →  0
fibonacci(1)  →  1
fibonacci(2)  →  1
```

Abbildung 4.1: Aufgabenstellung aus dem `CodingBat`-Datensatz

wählten Aufgaben aus den verschiedenen Datensätzen, sowie deren Repräsentation in den Studierendensequenzen. Zudem wird deutlich, welche Programmierkonzepte adressiert werden.

Die Auswahl der drei Aufgaben orientierte sich an ihrem Potenzial, typische Lernhürden und Feedbackanlässe des frühen Programmierlernens sichtbar zu machen.

`Fibonacci` (Abbildung 4.1) erfordert das Verständnis rekursiver Strukturen – ein Konzept, das zu den größten Herausforderungen beim Programmierenlernen zählt (Baron und Feitelson 2024; Fellah und Bandi 2018). Die Aufgabe bietet daher ein geeignetes Umfeld, um Feedbackentscheidungen im Hinblick auf typische Schwierigkeiten beim Verständnis und bei der Anwendung von Rekursion zu analysieren.

`MaxDigit` (Abbildung 4.2) verlangt den Einsatz grundlegender Kontrollstrukturen und Zeichenkettenverarbeitung. Die Aufgabenstellung ist sprachlich einfach, weist jedoch eine hinreichende logische Komplexität auf, um unterschiedliche Lösungsstrategien sowie typische Fehlerbilder hervorzurufen.

`Password` (Abbildung 4.3) fokussiert auf Benutzereingaben und Kontrollfluss (z. B. Wiederholungen, Bedingungen). Dabei können Situationen entstehen, in denen Lernende wiederholt mit Eingaben, Rückmeldungen und Zustandsänderungen konfrontiert sind. Die Aufgabe eignet sich deshalb insbesondere für die Analyse von Feedback in Kontexten, in denen die Interaktion mit dem System selbst ein zentrales Element darstellt – ein Aspekt, der in den anderen Aufgaben weniger ausgeprägt ist.

Max digit

Gegeben sei ein String, welcher nur Ziffern enthält.
Suche die größte Ziffer der Zeichenkette, und schreibe diese auf die Konsole.
Eingabe: Das Programm erhält eine Zeichenkette, die nur Ziffern enthält.
Ausgabe. Schreibe diejenige Ziffer auf die Konsole, die am höchsten ist und am häufigsten in der Zeichenkette vorkommt.

11111111	→	1
12309	→	9
4	→	4

Abbildung 4.2: Aufgabenstellung aus dem `TaskTracker`-Datensatz

Passwort

Schreiben Sie eine Funktion mit genau einem Parameter askPassword. Wenn die Funktion aufgerufen wird, fragt sie den Benutzer nach einem Passwort, bis der Benutzer das richtige Passwort eingibt. Das richtige Passwort wird der Funktion als Parameter übergeben. Wenn der Benutzer das richtige Passwort eingibt (d. h. die Eingabe des Benutzers entspricht dem als Parameter übergebenen Wert), gibt das Programm die Meldung 'Thank you!' aus.
Um ein Passwort abzufragen, verwenden Sie die Zeichenkette 'Enter password.'

Nachfolgend ist ein Beispiel der Funktion, wenn die Funktion in der Form askPassword('turnip') aufgerufen wird.

```
Enter password.
< cauliflower

Enter password.
< carrot

Enter password.
< turnip
Thank you!
```

Abbildung 4.3: Aufgabenstellung aus dem `FITech`-Datensatz

Die Auswahl der Programmiersprachen `Java`, `Dart` und `Python` erfolgte mit dem Ziel, eine Bandbreite an sprachspezifischen Herausforderungen abzubilden, die sich unmittelbar auf das Auftreten typischer Fehlermuster und damit auf potenzielle Feedbacksituationen auswirken. Jede der drei Sprachen bringt unterschiedliche syntaktische und semantische Eigenheiten mit sich, die jeweils charakteristische Fehlerquellen begünstigen können:

In `Python` etwa ist die Einrückung (sog. *Indentation*) nicht nur stilistisch, sondern syntaktisch relevant – eine Besonderheit, die bei Lernenden potenziell zu schwer nachvollziehbaren Fehlermeldungen führen kann, etwa wenn Codeblöcke unklar strukturiert sind (Denny et al. 2021). In `Java` hingegen liegt die Komplexität häufig in der strikten Typisierung, der Notwendigkeit expliziter Klassendefinitionen und der Arbeit mit geklammerten Codeblöcken, was andere Arten von Fehlerbildern sowie potenziell Missverständnisse hervorbringen kann (Tahsin, Fuad und Satter 2023). `Dart` wiederum vereint Elemente aus verschiedenen Paradigmen (z. B. objektorientiert, funktional) und wurde insbesondere aufgrund der Online-IDE ausgewählt, in der die zugehörige Aufgabe bearbeitet wurde. Diese Umgebung bot im Vergleich zu den anderen Systemen ein deutlich umfangreicheres Set an automatisierten Rückmeldungen und Hilfestellungen – darunter Inline-Hinweise, Syntaxmarkierungen, Fehlermeldungen sowie Formatierungsvorschläge. Für die Untersuchung eröffnet dies die Möglichkeit zu analysieren, inwieweit erfahrene Lehrpersonen diese systemseitigen Hinweise bei ihren Feedbackentscheidungen berücksichtigen oder gezielt aufgreifen. Die Verbindung von Programmiersprache und reichhaltigem Feedbackkontext stellt damit einen wichtigen Kontrastpunkt zu den anderen Bearbeitungsumgebungen dar. `Dart` wurde trotz seiner geringeren Verbreitung im Bildungskontext in die Untersuchung einbezogen, da seine syntaktische Nähe zu Sprachen wie `C#` erwarten lässt, dass erfahrene Lehrpersonen die Bearbeitungen auch ohne spezifische Vorkenntnisse in `Dart` nachvollziehen können.

4.2.1.3 Aufbereitung der Datensätze

Um die Bearbeitungsschritte der Lernenden im Rahmen eines Fragebogens darstellbar und für Lehrpersonen annotierbar zu machen, war es erforderlich, die einzelnen *Steps* in einer geeigneten Granularität aufzubereiten. In der Literatur werden sieben Ebenen der Granularität von Protokolldaten unterschieden (vgl. (Ihantola et al. 2015; Jeuring et al. 2022) und Tabelle 4.3).

Ebene	Beschreibung
`keystroke`	Einzelne Tastatureingaben
`token`	Spracheinheiten wie Schlüsselwörter oder Operatoren
`line`	Ganze Quelltextzeilen
`file-save`	Speichervorgänge einer Datei
`compile`	Kompiliervorgänge
`execute`	Programmausführungen
`submit`	Abgaben auf Plattformen (z. B. Online-Judge, LMS)

Tabelle 4.3: Ebenen der Granularität von Protokolldaten

Feinere Granularitäten – etwa `keystroke` – erlauben es, den Entstehungsprozess von Programmcode sehr detailliert nachzuvollziehen, sodass auch kleinste Bearbeitungsschritte, Korrekturen oder Umwege sichtbar werden und in eine Analyse einbezogen werden können. Mit zunehmender Detailtiefe steigt jedoch auch die Komplexität der Daten: Protokolle werden umfangreicher, sind schwerer manuell auszuwerten und für Menschen oft weniger intuitiv nachvollziehbar.

Grobgranulare Ebenen – etwa `execute` oder `submit` – reduzieren die Datenmenge erheblich und fassen den Bearbeitungsprozess in größeren Abschnitten zusammen. Dies erleichtert eine schnelle Übersicht und macht eine Annotation z. B. durch erfahrene Lehrpersonen praktikabler. Gleichzeitig gehen jedoch Zwischenschritte verloren, sodass Hinweise auf Fehlersuchen, Umwege oder kleinste Handlungen der Lernenden potenziell ausgeblendet werden.

Die Wahl der Granularität folgt der inhaltlichen Ausrichtung der Untersuchung und orientiert sich unmittelbar an der zugrunde liegenden Forschungsfrage. Entscheidend war, die Bearbeitungsschritte so zu erfassen, dass die *Entscheidungsprozesse von Lehrpersonen* beim Geben oder Unterlassen von Feedback nachvollziehbar analysiert werden können. Entsprechend musste die Darstellung der Bearbeitungsschritte ein Maß an Kontext bieten, das typische Interventionspunkte wie z. B. stagnierende Phasen, wiederkehrende Fehler oder das Erreichen teilrichtiger Zwischenlösungen erkennen lässt, zugleich aber hinreichend detailreich bleiben, um Veränderungen im Lösungsverhalten und deren mögliche Auslöser nachvollziehbar abzubilden. Vor diesem Hintergrund wurde die *Token-Ebene* gewählt: Sie erlaubt eine differenzierte, aber noch interpretierbare Sicht auf den Bearbeitungsprozess und bildet damit die Grundlage, um Entscheidungsprozesse von Lehrpersonen beim Geben oder Unterlassen von Feedback empirisch zu rekonstruieren. Damit bietet sie eine geeignete Grundlage, um situative Entscheidungen empirisch zu rekonstruieren und die Forschungsfrage nach den Bedingungen und Formen von Feedback im Programmierprozess systematisch zu bearbeiten. Zudem hat sich diese Granularität in früheren Untersuchungen als praktikabel erwiesen (Kiesler, Goethe-Universität Frankfurt Am Main und Hochschule Fulda 2022; Jeuring et al. 2022).

Die Aufbereitung der Datensätze erfolgte in Abhängigkeit von der jeweils verfügbaren Granularität. Während die Datensätze `CodingBat` und `FITech` bereits in vorverarbeiteter Form auf Token-Ebene vorlagen und ohne weitere Bearbeitung übernommen werden konnten, enthielt der `TaskTracker`-Datensatz Bearbeitungsverläufe in Keystroke-Granularität. Für jeden Tastendruck war ein neuer Codezustand protokolliert, was in Sequenzen mit mehreren Tausend Schritten resultierte. Für die Aufbereitung wurde das von Jeuring et al. (2022) beschriebene Verfahren verwendet. Dabei kam ein Skript zum Einsatz, das die Daten automatisiert komprimiert: Codezustände, in denen Lernende an derselben Code-Zeile arbeiten, werden zusammengefasst, wobei jeweils nur der letzte Zustand beibehalten wird. Zusätzlich wurden Schritte mit Benutzeraktionen (z. B. Ausführen des Codes) sowie Schritte vor und nach Pausen von mindestens zwei Minuten erhalten (für eine detaillierter Beschreibung siehe (Jeuring et al. 2022)).

4.2.1.4 Auswahl relevanter Steps

Obwohl der `FITech`-Datensatz bereits in passender Granularität verfügbar war, umfassten die einzelnen Sequenzen im Durchschnitt noch 144,1 *Steps* mit einer Standardabweichung von 120,7 *Steps*. Besonders lang waren dabei Sequenzen die zahlreiche Fehler im Code enthielten oder Sequenzen von Lernenden, die regelmäßig Feedback aus der Lernumgebung anforderten. Um den zeitlichen Aufwand für die teilnehmenden Lehrpersonen in einem vertretbaren Rahmen zu halten, wurde eine Obergrenze von 25 *Steps* pro Sequenz festgelegt. Erste Pilotdurchläufe zeigten, dass die Annotation eines einzelnen *Steps* im Mittel etwa ein bis zwei Minuten in Anspruch nimmt – abhängig von Komplexität und Kontext. Bei 25 *Steps* resultiert damit ein Bearbeitungszeitraum von etwa 30-45 Minuten pro Sequenz. Die Begrenzung soll sicherstellen, dass die vollständige Annotation ohne unverhältnismäßigen Zeitaufwand möglich ist und vorzeitige Abbrüche aufgrund zu langer Bearbeitungszeiten vermieden werden. Um die Länge der Sequenzen auf maximal 25 Bearbeitungsschritte zu reduzieren, wurden einzelne *Steps* entfernt, sofern sie nicht zum strukturellen Verständnis des Bearbeitungsverlaufs beitrugen. Dabei lag der Fokus darauf, *Steps* zu erhalten, die Aufschluss über das Vorgehen der Lernenden, das Auftreten potenzieller Fehlermuster oder typische Interaktionsmomente mit der Lernumgebung geben.

Das Entfernen einzelner Bearbeitungsschritte birgt grundsätzlich die Gefahr, dass feinere Nuancen des Bearbeitungsprozesses verloren gehen – etwa wiederholte Korrekturversuche, kleinere Tippfehler oder „kurze Irrwege“, die nicht unmittelbar zur Lösung beitragen, aber dennoch das Verhalten der Lernenden illustrieren könnten. Den potenziellen Nachteilen einer Reduktion stehen jedoch wesentliche Vorteile gegenüber: Die Sequenzen bleiben für Lehrpersonen überschaubar, die Vergleichbarkeit zwischen unterschiedlichen Bearbeitungsverläufen wird verbessert, und die Fokussierung auf zentrale Schritte unterstützt qualitativ fundierte Rückmeldungen. Auf diese Weise führt die Verdichtung zwar zu weniger, zugleich aber zu aussagekräftigeren Annotationen, die den Kern des Feedbackgeschehens präziser erfassen.

Um die genannten Einschränkungen methodisch zu adressieren, wurden klare *Include*- und *Exclude*-Kriterien definiert, die sich an den Vorarbeiten von Jeuring et al. (2022) orientieren und im Folgenden dargestellt werden. Die Kriterien gewährleisten, dass die Auswahl der *Steps* nachvollziehbar und replizierbar bleibt und zugleich eine konsistente Grundlage für künftige Studien schafft.

INCLUDE-Kriterien *Steps der Sequenz wurden aus folgenden Gründen beibehalten:*

- **Start-Code**, der von der Lernumgebung bereitgestellt wird.
  Der initiale Code bildet den Ausgangspunkt der Bearbeitung und ermöglicht, nachfolgende Änderungen im Kontext besser nachzuvollziehen.
- **Ausführen** oder **Kompilieren** sowie **Anfordern von Feedback**.
  Diese Aktionen markieren Stellen, an denen Lernende ihren aktuellen Stand überprüfen oder aktiv Unterstützung suchen. An diesen Punkten treten häufig automatisierte Rückmeldung durch das System auf, das für die Interpretation des weiteren Verlaufs bedeutsam sein kann.
- Einfügen von **kopiertem Code**.
  Das Einfügen von Code kann darauf hindeuten, dass Lernende Programmteile nicht selbst erstellt, sondern aus einer anderen Quelle übernommen haben. Für Lehrpersonen ist dies potenziell relevant, da ein solcher abrupter Fortschritt im Bearbeitungsverlauf anders interpretiert werden kann als kontinuierlich selbst geschriebener Code.
- Schritte direkt vor und nach einer **Pause von zwei Minuten oder länger**.
  Pausen unterbrechen den Lösungsfluss und können Anzeichen für Unsicherheit, Ablenkung oder bewusste Reflexion sein. Die nachfolgenden Schritte helfen dabei, die Funktion solcher Unterbrechungen im Bearbeitungsverlauf einzuordnen.
- Vollständige Codezeilen bei **durchgehender Eingabe**.
  Vollständige Codezeilen können auf abgeschlossene gedankliche Einheiten oder vollendete Planungsschritte der Lernenden hinweisen. Sie markieren damit mögliche Übergänge zwischen einzelnen kognitiven Arbeitsschritten. Zwischenschritte wurden nur dann beibehalten, wenn sie über die reine Eingabe hinaus Rückschlüsse auf potenzielle Denkprozesse oder Veränderungen im Vorgehen erlauben.
- **Fehler in Schlüsselwörtern oder Formatierungen**
  Der Step mit dem erstmaligen Auftreten dokumentiert den Ursprung des Fehlers. Bei Formatierungsfehlern wurden zusätzlich die davor- und danachliegenden Schritte beibehalten, um den Korrekturkontext sichtbar zu machen.

EXCLUDE-Kriterien *Steps der Sequenz wurden aus folgenden Gründen entfernt:*

- **Fortlaufendes Schreiben ohne Unterbrechung**.
  Wenn Lernende kontinuierlich Code schreiben – ohne Pausen, Löschvorgänge oder erkennbare Strukturwechsel – werden Zwischenschritte ausgelassen. Beibehalten wird nur ein *Step* pro vollständiger Codezeile

(vgl. INCLUDE-Kriterien), da die Zwischenzustände in der Regel keinen zusätzlichen Einblick in den Bearbeitungs- oder Denkprozess bieten.

- **Identische Steps mit gleichem Zeitstempel**.
  Wiederholte Steps mit identischem Zeitstempel gelten als Duplikate und werden entfernt, da sie keine inhaltliche Veränderung enthalten und daher keinen Beitrag zur Analyse leisten.

- **Mehrfaches Kompilieren ohne Codeänderung**.
  Wenn ein Programm mehrfach hintereinander ausgeführt wird, ohne dass sich der Code dazwischen verändert, werden nur zwei Ausführung beibehalten. Weitere Wiederholungen liefern in solchen Fällen keine zusätzlichen Informationen über den Bearbeitungsverlauf oder die kognitiven Prozesse der Lernenden.

Die Anwendung der beschriebenen Kriterien führte zu einer kompakten Auswahl annotierbarer Bearbeitungsverläufe, die sich hinsichtlich Länge, Aufgabenstellung und Datensatzzugehörigkeit unterscheiden. Tabelle 4.4 gibt einen Überblick über die finalen Sequenzen, einschließlich Quelle, Aufgaben-ID im Datensatz, Aufgabenname[3] und Anzahl der enthaltenen *Steps*.

Quelle	ID	Name	Aufgabe	Steps
`CodingBat`	B02	Bob	Fibonacci	9
`CodingBat`	B05	Alice	Fibonacci	10
`FITech`	148	Eve	Password	23
`FITech`	538	Dave	Password	15
`TaskTracker`	tt1	Parker	MaxDigit	12
`TaskTracker`	tt2	Jasmine	MaxDigit	22

Tabelle 4.4: Übersicht über die ausgewählten Sequenzen

4.2.1.5 Nachbildung der Steps

Um eine möglichst realitätsnahe Darstellung der Bearbeitungssituationen zu gewährleisten, wurden die finalen Sequenzen mit den ausgewählten *Steps* in den Lernumgebungen, in denen die Daten aufgezeichnet wurden, nachgestellt. Ziel war es, den Teilnehmenden exakt jene Darstellungsform zugänglich zu machen, die auch die Lernenden im ursprünglichen Bearbeitungskontext wahrgenommen hatten – einschließlich aller visuellen, funktionalen und inhaltlichen Systemreaktionen. Auf diese Weise soll eine kontextgerechte Einschätzung durch die Teilnehmenden ermöglicht und zugleich sichergestellt werden, dass auch situative Aspekte sowie systembedingte Rückmeldungen als potenziell relevante Faktoren in die Feedbackentscheidungen einbezogen werden können.

Dazu gehören:

- die originalgetreue Anzeige von Syntax-Highlighting, Farbkodierungen und Formatierungen,
- die Wiedergabe der vom System bereitgestellten Rückmeldungen (z. B. Compiler-Fehlermeldungen, Hinweise)
- die Darstellung plattformspezifischer Funktionsweisen (z. B. Unterschiede in Rückmeldungen bei Codeausführungen)

Für jeden *Step* wurde der zugehörige Quellcode manuell in der jeweiligen Lernplattform rekonstruiert. Benutzeraktionen im ursprünglichen Bearbeitungsverlauf, wie etwa `RUN`, `HELP` oder

[3]Der Aufgabenname entspricht den Namen der Personas, welche in Abschnitt 4.2.2.1 weiter erläutert werden

`SUBMIT`, wurden ebenfalls im entsprechenden *Step* nachgebildet. Anschließend wurde ein Screenshot des relevanten Bildausschnitts der Lernumgebung erstellt, sodass der aktuelle Programmcode zusammen mit möglichen Systemmeldungen oder Hilfetexten für die Lehrpersonen sichtbar war. Auf diese Weise konnte sichergestellt werden, dass den Teilnehmenden sämtliche für die Feedbackbewertung potenziell relevanten Informationen vollständig und nachvollziehbar vorlagen. Abbildung 4.4 zeigt exemplarisch eine solche Rekonstruktion eines Bearbeitungsschrittes – hier *Step* 5 aus dem `FITech`-Datensatz.

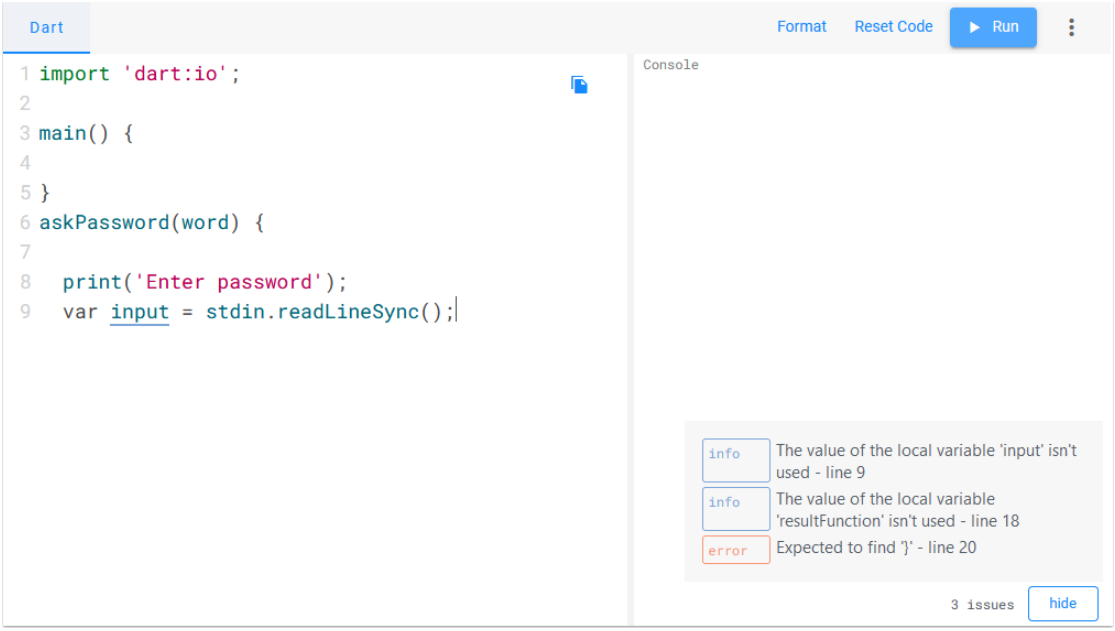


Abbildung 4.4: Screenshot der Nachbildung von *Step* 5 (Dave) in der Lernumgebung `FITech`

Die beschriebenen Auswahl- und Aufbereitungsschritte führten zu einer finalen Auswahl von sechs Bearbeitungsverläufen, die entweder vollständig oder in gekürzter Form charakteristische Abschnitte aus den Lösungsprozessen zu drei unterschiedlichen Programmieraufgaben enthalten. Sie basieren auf realen Interaktionen von Programmierlernenden mit digitalen Lernumgebungen und spiegeln eine heterogene Bandbreite typischer Bearbeitungsmuster, Fehlerverläufe und Interaktionsformen wider.

Die drei Aufgaben stammen dabei jeweils aus unterschiedlichen Kontexten und wurden in verschiedenen Programmiersprachen bearbeitet (`Java`, `Dart`, `Python`). Dadurch ergibt sich eine inhaltlich wie technisch vielfältige Grundlage für die Untersuchung von Feedbackentscheidungen.

Sämtliche für die Analyse verwendeten Sequenzen sind zudem öffentlich zugänglich unter `https://github.com/Programming-Steps-Working-Group-2022/Expert-Reasons-ITiCSE-24`.

4.2.2 Umfrage zur Erhebung von Feedbackentscheidungen

Um systematisch Feedback-Entscheidungen erfahrener Lehrpersonen zu konkreten Bearbeitungssituationen zu erfassen und damit einen empirischen Beitrag zur Beantwortung der in Abschnitt 3.2 formulierten Forschungsfragen zu leisten, wurde auf Grundlage der in den vorangegangenen Abschnitten beschriebenen Bearbeitungsverläufe eine Online-Umfrage konzipiert. Die nachfolgenden Abschnitte erläutern die Konzeption des Fragebogens, die Durchführung der Erhebung sowie die Zusammensetzung der Stichprobe.

4.2.2.1 Konzeption des Fragebogens

Der Fragebogen gliedert sich in zwei Abschnitte: einen *demographischen* Teil und einen *aufgabenbezogenen* Teil. Diese Trennung diente dazu, zunächst zentrale Angaben zur Person und Erfahrung der Teilnehmenden zu erfassen und im Anschluss deren konkrete Entscheidungen zu ausgewählten Bearbeitungssequenzen zu erheben.

Im **ersten Teil** des Fragebogens werden grundlegende Angaben zur Person und zur beruflichen Erfahrung der Teilnehmenden erhoben, darunter Berufsbezeichnung, Unterrichtserfahrung im Programmieren (in Jahren), Bildungsstufe sowie das Land, in dem überwiegend unterrichtet wird. Diese Informationen dienen in erster Linie der Charakterisierung der Stichprobe sowie der Überprüfung, ob die Teilnehmenden den für die Studie definierten Kriterien erfahrener Lehrpersonen entsprechen. Die Definition von „erfahrenen Lehrpersonen“ bezieht sich dabei primär auf Praxiserfahrung in der Programmierlehre, unabhängig vom konkreten Bildungskontext oder der formalen Berufsbezeichnung. Die Unterrichtserfahrung im Programmieren dient hierbei, um sicherzustellen, dass ausschließlich Personen mit einschlägiger Lehrerfahrung in die Auswertung einbezogen werden. Weitere demografische Daten wie Geschlecht, Alter oder Ausbildung werden bewusst nicht erhoben, da der Fokus der Studie auf den Entscheidungsprozessen beim Geben von Feedback liegt und nicht auf der Analyse individueller Personenmerkmale. Durch eine gezielte Verteilung des Fragebogens (siehe Abschnitt 4.2.2.2) soll jedoch gewährleistet werden, dass die Stichprobe Diversität in zentralen Merkmalen wie Geschlecht einschließt.
Zu Beginn werden die Teilnehmenden gebeten anzugeben, in welchen der für die Studie relevanten Programmiersprachen sie sich beim Geben von Feedback als kompetent einschätzen. Damit soll verhindert werden, dass Sequenzen in einer Sprache annotiert werden, deren Syntax oder Besonderheiten den Teilnehmenden nicht vertraut ist. Somit soll vermieden werden, dass frühzeitige Abbrüche aufgrund mangelnder Vertrautheit mit der verwendeten Programmiersprache auftreten. Eine besondere Herausforderung stellen hierbei die Sequenzen in `Dart` dar, da diese Programmiersprache im internationalen Bildungskontext bislang nur eine geringe Verbreitung aufweist und daher vielen potenziellen Teilnehmenden weniger vertraut sein dürfte. Da sich `Dart` im konkreten Aufgaben- und Interaktionskontext jedoch nur geringfügig von bekannten Programmiersprachen wie `Java` oder `C#` unterscheidet, wird in dieser Studie die Annahme getroffen, dass Lehrpersonen mit Erfahrung in einer dieser beiden Sprachen auch in der Lage sind, fundiertes Feedback zu `Dart`-Code zu geben. Entsprechend wird `Dart` in der Sprachzuweisung den `Java`- und `C#`-Erfahrenen Lehrpersonen zugeordnet. In der Sprachabfrage wird bewusst nicht nach `Dart` gefragt, da anzunehmen ist, dass die Selbsteinschätzung in dieser wenig verbreiteten Sprache zu zurückhaltend ausfallen würde. Stattdessen wird `C#` abgefragt und im Sinne der genannten Annahme als ausreichend für die Bearbeitung von `Dart`-Sequenzen betrachtet. Ein möglicher Einfluss dieser Entscheidung wird in Abschnitt 4.6 diskutiert.

Im zweiten Teil des Fragebogens – dem **aufgabenbezogenen Teil** – erhalten die Teilnehmenden unter Berücksichtigung ihrer zuvor angegebenen Programmiersprachenpräferenzen zufällig eine der sechs vorbereiteten Bearbeitungssequenzen. Jede dieser Sequenzen wird durch zwei zentrale Elemente eingeführt: (1) eine kompakte **Aufgabenbeschreibung** und (2) eine **Persona**, die die Bearbeitungssituation kontextualisiert. Die Aufgabenbeschreibung gibt den Teilnehmenden einen vollständigen Überblick über das zu lösende Programmierproblem, wie es auch Lernenden in der jeweiligen Lernumgebung vorliegt. Sie bildet den inhaltlichen Bezugsrahmen für die nachfolgend präsentierten Bearbeitungsschritte und ermöglicht es den Teilnehmenden, ihre Rückmeldungen in Bezug auf die Aufgabenstellung und den intendierten Lösungsweg einzuordnen. Um auch individuelle Faktoren in den Feedbackentscheidungen berücksichtigen zu können, wird für jede der sechs Bearbeitungssequenzen zusätzlich eine **Persona** bereitgestellt, die die Bearbeitungssituation kontextualisiert und relevante Hintergrundmerkmale der lernenden Person sichtbar macht. Die Personas basieren dabei auf Informationen aus den ursprünglichen Datensätzen (z. B. Al-

tersgruppe, Nutzungskontext, Lernumgebung) und repräsentieren unterschiedliche individuelle Merkmale, Zielorientierungen und lernbezogene Voraussetzungen der Programmierenden. Die Darstellung der jeweiligen Persona – einschließlich Name, Alter, Zielsetzung und schulischer bzw. beruflicher Rolle – hat dabei das Ziel, neben situativen Faktoren auch die individuellen Voraussetzungen der Lernenden in die Beurteilung von Feedback-Entscheidugnen einbeziehen zu können. Die visuelle Darstellung durch einen Avatar soll dabei die Identifikation mit der jeweiligen Persona unterstützen und es den Teilnehmenden ermöglichen, Rückmeldungen stärker in einem konkreten sozialen und individuellen Kontext zu verorten. Abbildung 4.5 zeigt beispielhaft die Persona-Beschreibung von Dave und Jasmine, wie sie im Fragebogen präsentiert wird. Eine Übersicht aller Personas mit Kurzbeschreibungen findet sich in Tabelle 4.6. Die vollständigen Persona-Beschreibungen in Originalsprache (Englisch), wie sie in der Umfrage eingesetzt werden – einschließlich Aufgaben-IDs und zugehöriger Profilbilder – sind im Anhang A.1 dokumentiert.

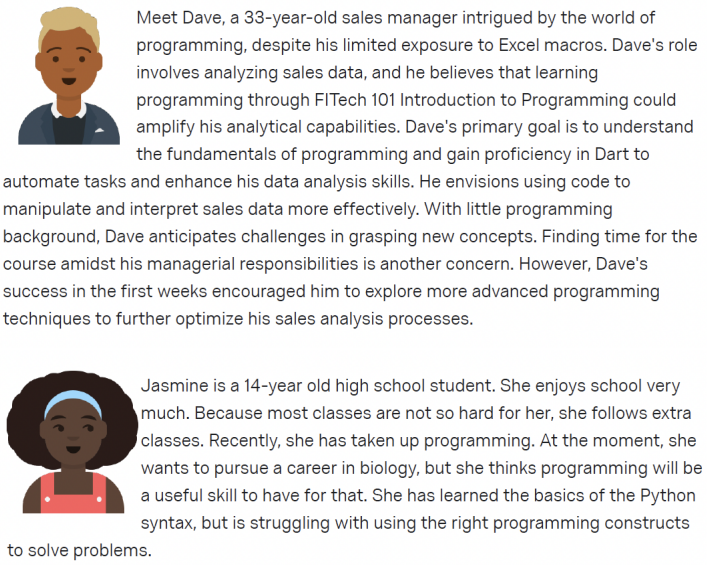


Abbildung 4.5: Screenshot mit Beschreibung und Avatar der Persona `Dave` und `Jasmine` aus dem Fragebogen

Im Anschluss an die Einführung durch Aufgabenbeschreibung und Persona werden den Teilnehmenden die Bearbeitungsschritte der Lernenden aus der jeweils ausgewählten Sequenz nacheinander präsentiert. Für jeden einzelnen *Step* werden drei Items erhoben, die zentrale Aspekte der Forschungsfragen adressieren. Sie erfassen *ob* sich für eine Rückmeldung entschieden wird, das *warum* der jeweiligen Entscheidung sowie das *wie* der konkreten Ausgestaltung des Feedbacks. Die originalen Formulierungen sowie das zugehörige Antwortformat sind in Tabelle 4.5 zu finden.

Item-ID	Formulierung	Antwortmöglichkeit
`IF`	*Would you give feedback at this point?*	*Yes / Maybe / No*
`WHY`	*Why?*	Offene Freitextantwort
`HOW`	*Which feedback?*	Offene Freitextantwort

Tabelle 4.5: Übersicht der Items im aufgabenbezogenen Teil des Fragebogens

Name	Alter	Beschreibung
Bob	31	Informatikstudent an einer Hochschule für angewandte Wissenschaften in Deutschland, nutzt Programmieren zur Problemlösung und Automatisierung im Alltag. Hat solide Vorkenntnisse.
Alice	19	Angehende App-Entwicklerin mit guten Programmierkenntnissen, nimmt aus Neugier an einem Usability-Experiment teil.
Eve	30	Marketing-Fachkraft, will grundlegendes Programmierverständnis aufbauen, um besser mit Entwicklerteams kommunizieren zu können.
Dave	33	Vertriebsleiter, möchte seine Datenanalysefähigkeiten durch Programmieren erweitern. Hat erste Fortschritte gemacht, ist aber beruflich stark eingebunden.
Parker	16	Schüler mit Programmierinteresse durch die Mutter (Softwareentwicklerin). Begeistert, aber etwas überfordert mit professioneller Entwicklungsumgebung.
Jasmine	14	Vielseitig interessierte Schülerin mit ersten Python-Kenntnissen, die Programmieren als Zusatzqualifikation für ihre angestrebte Biologiekarriere sieht.

Tabelle 4.6: Kurzbeschreibungen der verwendeten Personas

Die Formulierungen der Items sind bewusst knapp gehalten, um die kognitive Belastung der Teilnehmenden gering zu halten, Missverständnisse zu vermeiden und eine konsistente Beantwortung über eine Vielzahl von *Steps* hinweg zu ermöglichen. Auf diese Weise soll sichergestellt werden, dass der Fokus auf den inhaltlichen Entscheidungen liegt und nicht durch komplexe oder interpretativ offene Fragestellungen überlagert wird.

Bei der Konzeption des Fragebogens wurden mehrere Gestaltungsprinzipien bewusst implementiert, um eine realitätsnahe und zugleich methodisch stringente Erfassung der Feedbackentscheidungen zu ermöglichen:

- **Keine Rücksprungmöglichkeit**: Um zu verhindern, dass Teilnehmende ihre Entscheidungen rückblickend auf Basis späterer Schritte revidieren oder sich zunächst alle *Steps* nacheinander ansehen, bevor sie sich für einen Zeitpunkt und die Art des Feedback entscheiden – ein Vorgehen, das im realen Unterricht nicht möglich ist – wurde eine Navigationsmöglichkeit zu vorherigen *Steps* explizit ausgeschlossen.
- **Darstellung zweier aufeinanderfolgender Zustände**: Da es keine Möglichkeit gab, innerhalb der Sequenz zwischen den einzelnen *Steps* frei zu navigieren, wurden jeweils zwei direkt aufeinanderfolgende Codezustände angezeigt: der zuvor bereits dargestellte Bearbeitungsschritt *Previous Step* gefolgt vom aktuellen zu annotierenden *Actual Step*. Dies sollte den Lernfortschritt besser nachvollziehbar machen und die sequentielle Einordnung des aktuellen Bearbeitungsschritts erleichtern.
- **Jeder *Step* ist isoliert zu bewerten**: Da es sich bei den gezeigten Bearbeitungsverläufen um aufgezeichnete, feststehende Sequenzen handelt, hat das im Rahmen des Fragebogen durch die Lehrpersonen angegebene Feedback keinen Einfluss auf nachfolgende *Steps*. Um Missverständnissen vorzubeugen, wurden die Teilnehmenden daher explizit angehalten, jeden *Step* unabhängig von zuvor gegebenem Feedback zu beurteilen – also so, als hätten die Lernenden bisher keinerlei Feedback der Lehrperson erhalten. So sollte verhindert werden, dass Rückmeldungen „in Gedanken“ weiterwirken und spätere Einschätzungen beeinflussen.
- **Kein wiederholtes Feedback bei fortbestehenden Fehlern**: Um Dopplungen zu vermeiden, wurden die Teilnehmenden angewiesen, zu einem bestimmten Fehler nur einmal

Feedback zu geben – auch wenn dieser über mehrere Schritte hinweg bestehen blieb. Andernfalls wäre unklar, ob wiederholte Rückmeldungen bewusst oder versehentlich erfolgen, was die nachfolgende Interpretation bei der Analyse erschweren würde.

Die genannten Prinzipien werden den Teilnehmenden zu Beginn des aufgabenbezogenen Teils ausdrücklich in der Instruktion des Fragebogens vermittelt. Der folgende Auszug zeigt exemplarisch die Formulierung für die Sequenz *Bob* und verdeutlicht, wie die Vorgaben sprachlich umgesetzt wurden:

> *In the next couple of questions we will ask what kind of feedback you would give to Bob, who is working on the above task. In each question, we show two consecutive states of the program Bob is developing. The states are screenshots of the online programming environment in which he works, and also show the feedback from the environment. Each time we will ask whether or not you would give feedback to Bob after the step, and if so, why and what. We included a step if Bob added, removed, or changed a line, pressed the Run button, or paused for more than 2 minutes.*
>
> *As we want to simulate a classroom setting, we will show you Bob's progress going forward without the possibility to go back and change the feedback you provided for previous steps. Also, you may assume that Bob did not receive the feedback you provide, since in reality, Bob of course didn't get this feedback. Please give feedback only once for a particular error, and ignore the error in the steps that come later.*

Abbildung 4.6 zeigt exemplarisch zwei aufeinanderfolgende *Steps* aus der Sequenz *Dave*, in denen eine Codezeile zunächst vervollständigt und anschließend mittels `RUN`-Aktion ausgeführt wird. Um das Ausführen der Aktion den Lehrpersonen zu visualisieren, wurde die Schaltfläche `RUN` in der Darstellung hervorgehoben. Diese Art der Hervorhebung war in allen Sequenzen einheitlich gestaltet und wurde im Rahmen der Nachbildung der Bearbeitungsschritte entsprechend annotiert.

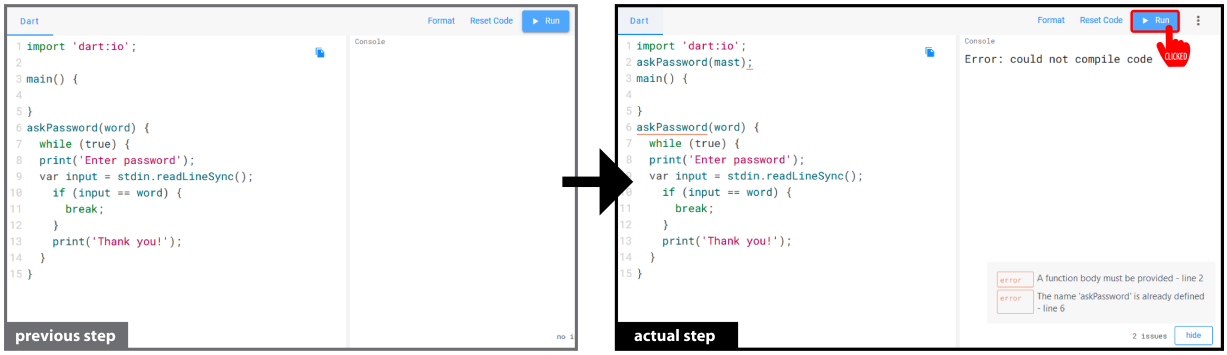


Abbildung 4.6: Zwei aufeinanderfolgende *Steps* von Dave; im aktuellen *Step* (rechts) wurde die Aktion `RUN` ausgelöst.

4.2.2.2 Rekrutierung und Verbreitung der Umfrage

Die Umfrage ist gezielt für erfahrene Lehrpersonen konzipiert, die Programmieren in unterschiedlichen Bildungskontexten unterrichten. Ziel ist es, eine möglichst vielfältige Perspektive auf reale Feedbackpraktiken zu erfassen und unterschiedliche Kontexte der Programmierausbildung abzubilden (z. B. schulisch vs. universitär, verschiedene Sprachen und Curricula).

Um dieses Ziel zu erreichen, wird ein mehrgleisiges Rekrutierungskonzept verfolgt, das auf eine international heterogene und unterrichtserfahrene Stichprobe abzielt.

Die Verbreitung erfolgt über mehrere Kanäle:

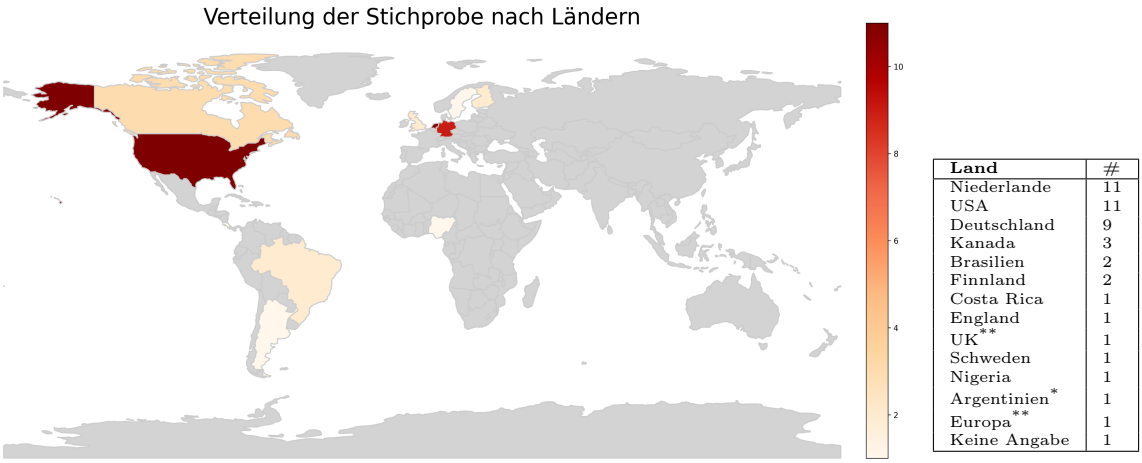
- internationale Fachmailinglisten, insbesondere die `SIGCSE`-Mailingliste der ACM,
- einschlägige wissenschaftliche Konferenzen – darunter `ITiCSE 2023`, `DELFI 2023` und `FIE 2023`,
- regionale und nationale Fachtreffen für Informatiklehrkräfte – insbesondere aus dem Sekundarbereich,
- gezielte Verbreitung über persönliche Netzwerke – insbesondere an erfahrene Hochschullehrende für Einführungskurse in der Programmierung sowie Lehrkräfte für Informatik an Schulen,
- Schneeballprinzip: Teilnehmende und Fachkolleginnen und -kollegen werden gebeten, den Fragebogen an geeignete Personen im eigenen Netzwerk weiterzuverbreiten.

Die Kombination aus offenen und gezielten Kanälen soll die Ansprache einer diversen Zielgruppe bei gleichzeitigem Fokus auf qualitativ hochwertiges Expertenwissen ermöglichen. Durch die internationale Ausrichtung wird der Anspruch der Studie gestützt, allgemeingültige Muster pädagogischer Feedback-Strategien zu identifizieren, ohne sich auf ein einzelnes Bildungssystem zu beschränken. Dabei können auch potenzielle Einflüsse kultureller Unterschiede berücksichtigt werden – etwa in Bezug auf den Umgang mit Fehlern, die Rolle der Lehrperson oder typische Formen von Rückmeldung in verschiedenen Lehrkontexten.

4.2.2.3 Beschreibung der Stichprobe

Hinweis: Alle Teilnehmenden gaben vor Beginn der Umfrage ihre Zustimmung zur Verwendung ihrer Daten für die vorliegende Untersuchung.

Insgesamt nahmen **47 Personen** an der Befragung teil. Die Teilnehmenden stammen aus unterschiedlichen Weltregionen, wobei der überwiegende Teil aus Europa ($n = 26$) und Nordamerika ($n = 14$) vertreten ist. Weitere Rückmeldungen wurden aus Mittel- und Südamerika sowie Afrika verzeichnet. Abbildung 4.7 zeigt eine Übersicht der Verteilung der Stichprobe auf die Herkunftsländer. Die Teilnehmenden verfügen über umfangreiche Erfahrung im Unterrichten von Programmierkursen: im Durchschnitt **13,3 Jahre** (Median: 8 Jahre). Die Standardabweichung betrug **11,65 Jahre**, was auf eine heterogene Verteilung der Erfahrung hindeutet.



Land	#
Niederlande	11
USA	11
Deutschland	9
Kanada	3
Brasilien	2
Finnland	2
Costa Rica	1
England	1
UK**	1
Schweden	1
Nigeria	1
Argentinien*	1
Europa**	1
Keine Angabe	1

* gab zusätzlich an, zeitweise in der Schweiz tätig zu sein. ** unspezifizierte Angabe.

Abbildung 4.7: Herkunftsländer der Teilnehmenden – visualisiert und tabellarisch

4.2.3 Gemeinsamer Auswertungsrahmen der Teilanalysen

Die Auswertung der erhobenen Daten orientiert sich unmittelbar an den in Abschnitt 3.2 formulierten Forschungsfragen und erfolgt entlang zweier zentraler Perspektiven:

- der Analyse der Entscheidungsgründe für oder gegen eine Rückmeldung (**FF1.1**)
  – nachfolgend: `WHY`-Analyse
- der Analyse der inhaltlichen Ausgestaltung der gegebenen Rückmeldungen (**FF1.2**)
  – nachfolgend: `HOW`-Analyse

Während die nachfolgenden Abschnitte zunächst das gemeinsame methodische Vorgehen beschreiben, werden die beiden Teilanalysen anschließend separat dargestellt. Dabei werden jeweils die analytischen Schwerpunkte, die zugrunde liegenden Kategoriensysteme sowie der spezifische Kodierprozess dargestellt.

4.2.3.1 Vorbereitung der Materialbasis

Vor Beginn der inhaltlichen Analyse wurden die erhobenen Daten systematisch bereinigt, um sicherzustellen, dass ausschließlich aussagekräftige und kodierbare Texte in die weitere Auswertung einfließen. Dieser Schritt ist im Sinne von Mayring (2015a) nicht als eigentliche Kodierung zu verstehen, sondern als vorbereitende Materialaufbereitung. Er dient der Sicherung einer klar abgegrenzten und inhaltlich belastbaren Analyseeinheit, die erst im Anschluss der regelgeleiteten Kategorienbildung und Kodierung zugeführt wird.

Hierfür wurden zunächst sämtliche offenen Antworten zu den Items (`WHY` und `HOW`) gesichtet und diejenigen markiert, die keine inhaltliche Substanz aufwiesen. Dazu zählen etwa leere Felder oder nicht interpretierbare Eingaben ohne erkennbaren Aussagegehalt. Für das Item `WHY` wurden solche Fälle beispielsweise mit dem formalen Label `NOR` (für „No Reason“) versehen, um sie systematisch kenntlich zu machen. Auf diese Weise bleibt nachvollziehbar dokumentiert, welche Antworten bewusst als nicht kodierbar eingestuft wurden. Zugleich wird so sichergestellt, dass der Ausschluss dieser Einträge transparent und replizierbar bleibt, ohne dass Rohdaten gelöscht oder unberücksichtigt bleiben. Die Vergabe solcher Labels stellt keine inhaltliche Kodierung dar, sondern ist als technische Vor-Kodierung zu verstehen, die die eigentliche Analyse vorbereitet. Antworten, die nicht eindeutig als irrelevant oder verwertbar eingestuft werden konnten, wurden in einer gemeinsamen Diskussion mit den Kodierenden besprochen und im Konsens entschieden.

4.2.3.2 Datenbasis der qualitativen Analysen

Die gesamte Datenbasis umfasst insgesamt 464 Antworten zu Item `WHY` sowie 210 Antworten zu Item `HOW`. Berücksichtigt werden ausschließlich bereinigte Antworten, die als inhaltlich aussagekräftig eingestuft sind und damit die Grundlage für die qualitative Inhaltsanalyse bilden (vgl. Abschnitt 4.2.3.1).

Die Anzahl der Antworten auf die beiden Items variiert zwischen den einzelnen Bearbeitungssequenzen und ist in Tabelle 4.7 dargestellt. Die geringere Zahl an Antworten zu Item `HOW` ergibt sich daraus, dass eine konkrete Feedbackformulierung in der Regel nur dann eingetragen wurde, wenn Teilnehmende zuvor im Rahmen von Item `IF` angegeben hatten, tatsächlich Feedback geben zu wollen. In allen anderen Fällen blieb das Textfeld meist ohne Inhalt.

4.2.3.3 Methodisches Vorgehen der Inhaltsanalyse

Die Auswertung der Daten in beiden Teilanalysen basierte auf der Methodik der qualitativen Inhaltsanalyse nach Mayring (2015a) (siehe Abschnitt 3.3.2).

Sequenz	WHY	HOW
Bob	38	40
Alice	76	33
Eve	177	42
Dave	37	25
Parker	102	45
Jasmine	34	25
Gesamt	**464**	**210**

Tabelle 4.7: Bereinigte Datenbasis: Verteilung der Antworten auf die Items `WHY` und `HOW`

Zunächst werden für beide Analysen deduktive Kategorien aus der Literatur übernommen, die auf relevante Studien in den jeweiligen Teilbereichen zurückgehen. Die Auswahl wird in den jeweils nachfolgenden Abschnitten begründet (vgl. Abschnitt 4.2.4.1 für die `WHY`-Analyse, Abschnitt 4.2.5.1 für die `HOW`-Analyse). Im weiteren Verlauf der Analyse werden diese Kategorien iterativ durch induktiv entwickelte Kategorien ergänzt und verfeinert, um domänenspezifische Besonderheiten sowie kontextspezifische Argumentationsmuster der befragten Lehrpersonen abzubilden.

Die Kodierung erfolgt in mehreren Runden durch ein Team von insgesamt vier Kodierenden (nachfolgend mit `C1` bis `C4` bezeichnet, wobei `C1` der Verfasser der Dissertation ist). Dieses Vorgehen dient einerseits der methodischen Absicherung – etwa durch Gegenlesen und Perspektivabgleich – und ermöglicht andererseits eine differenzierte Reflexion der Kategoriendefinitionen im interdisziplinären Team. Die Aufteilung in mehrere Durchläufe unterstützt zudem die schrittweise Entwicklung und Validierung des Kategorienschemas. Nach Abschluss der jeweiligen Kodierungsphasen werden offene Konflikte in gemeinsamen Sitzungen aller beteiligter Kodierender diskutiert und aufgelöst. Diese Sitzungen werden von `C1` geleitet, um den Prozess zu strukturieren und eine Konsensbildung zu sichern. Das Vorgehen folgt der qualitativen Inhaltsanalyse nach Mayring (2015a), die die theoretisch fundierte Konsensbildung als zentrales Element zur Sicherung von Interpretationskonsistenz hervorhebt. Ziel ist es, ein einheitliches Begriffsverständnis zu etablieren und die Zuverlässigkeit der Kodierungen über alle Sequenzen und beide Analyseteile hinweg zu erhöhen. Für zentrale Abschnitte soll die Intercoder-Zuverlässigkeit mit Cohens κ überprüft werden (Cohen 1960).

4.2.4 Analyse der Feedback-Entscheidungen (WHY-Analyse)

Zur Analyse der Beweggründe, aus denen erfahrene Lehrpersonen entscheiden, ob sie an einer bestimmten Stelle Feedback geben, wird ein strukturierter, mehrstufiger Kodierprozess durchgeführt (Mayring 2015a). Grundlage hierfür bilden die offenen Freitextantworten auf Item `WHY`, die jeweils eine zuvor im Rahmen von Item `IF` getroffene Entscheidung begründen. Diese Begründungen bilden das Korpus, das der `WHY`-Analyse zugrunde liegt.

4.2.4.1 Ausgangspunkt: Kategoriensystem und Erweiterung

Ausgangspunkt der Analyse ist das Kategoriensystem von Jeuring et al. (2022), das typische Rückmeldestrategien im Kontext textbasierter Programmieraufgaben systematisiert. Es wird als deduktives Gerüst für die erste Kodierphase herangezogen und bildet damit die Grundlage, um potenziell relevante Ereignisse im Bearbeitungsprozess zu identifizieren. Die vorliegende Untersuchung knüpft gezielt an diese Vorarbeiten an und erweitert das bestehende System um zusätzliche Kategorien, die sich induktiv aus dem Material ergeben.

Tabelle 4.8 gibt einen Überblick über das übernommene Kategoriensystem, das als Ausgangspunkt für die `WHY`-Analyse dient und somit die in den Antworten auf Item `WHY` begründeten Feedbackentscheidungen strukturiert.

Kategorie	Beispiel	Interventionspunkt
Compiler error	Syntax or type error	If a student is not using the compiler often, or after the second compilation where the error goes unaddressed.
Semantic error	Syntax is correct but wrong semantics, e.g. "=" instead of "=="	When a student moves to the next line, as these errors are hard to spot/debug.
Logical error		Once a student executes/tests the code or within 5 minutes if they choose not to run the code. If the error would lead to any further code being incorrect, intervene immediately.
Deviation from specification	Changing required function signature	When a student leaves the line.
Trial and Error behaviour	Iterating through conditional operands	Once it becomes clear the edits are guessing – not experimentation.
Hint or Feedback request	Pressing a "Hint" or "Show solution" button	Immediately when a student requests assistance.
Subgoal completion	Correct base case(s) for recursive function	When a student completes all steps of a subgoal.

Tabelle 4.8: Kategoriensystem nach Jeuring et al. (2022)

4.2.4.2 Kodierprozess: Ablauf und Entwicklung des Schemas

Die Entwicklung des Kategoriensystems für die `WHY`-Analyse erfolgt in einem regelgeleiteten, iterativen Verfahren im Sinne der qualitativen Inhaltsanalyse nach Mayring (2015a). Tabelle 4.9 zeigt die Zuordnung der Kodierenden zu den jeweiligen Datensätzen bzw. Sequenzen[4].
Der Ablauf lässt sich in folgende Schritte gliedern:

- **Erste Testkodierung**: Drei Kodierende wenden das deduktiv übernommene Kategoriensystem unabhängig voneinander auf eine erste Sequenz (*Alice*, 76 Antworten) an. Anschließend werden die Ergebnisse verglichen und in einer Konsenssitzung diskutiert, um ein gemeinsames Begriffsverständnis zu sichern und die Kategorien an das Material anzupassen.
- **Anwendung auf weiteres Material**: Die überarbeitete Kategoriestruktur wird auf eine zweite Sequenz (*Bob*, 38 Antworten) übertragen und auf ihre Tragfähigkeit geprüft. Dabei erfolgt eine erste Schärfung der Definitionen.
- **Erweiterung und Präzisierung**: Bei der Kodierung weiterer Sequenzen (*Dave*, 37 Antworten; *Eve*, 177 Antworten) werden neue Kategorien ergänzt und bestehende differenziert, sodass induktiv gewonnene Aspekte systematisch in das Kategoriensystem integriert werden.

[4]In der `WHY`-Analyse übernimmt C3 keine eigenen Kodierungen, wirkt jedoch aktiv an der Entwicklung und Präzisierung des Kategoriensystems mit und ist in die Konsenssitzungen eingebunden. Da C3 in der `HOW`-Analyse zentrale Kodieraufgaben übernimmt, wird die Person in der Tabelle dennoch aufgeführt, um die Übersicht konsistent zu halten und die Beteiligung aller vier Kodierenden gleichermaßen sichtbar zu machen.

- **Überprüfung der Reliabilität**: Zur methodischen Absicherung wird auf Grundlage der Sequenz *Parker* (102 Antworten) die Intercoder-Reliabilität mit Cohens κ berechnet.
- **Abschluss der Kodierung**: Mit der Kodierung der letzten Sequenz (*Jasmine*, 34 Antworten) wird das stabilisierte Kategoriensystem ohne weitere Anpassungen angewendet, womit der Kodierprozess abgeschlossen ist.

ID	CodingBat		FITech		JetBrains	
	Bob	**Alice**	**Eve**	**Dave**	**Parker**	**Jasmine**
C1	x	x	x	x	x	x
C2	x	x	x	x	x	x
C3						
C4	x	x				

Tabelle 4.9: Zuordnung der Kodierenden zu den jeweiligen Datensätzen (WHY-Analyse)

4.2.5 Analyse der Feedback-Typen (HOW-Analyse)

Analog zur WHY-Analyse (vgl. Abschnitt 4.2.4) wird zur Untersuchung der konkreten Ausgestaltung von Feedbacknachrichten ein regelgeleiteter, mehrstufiger Kodierprozess nach Mayring (2015a) eingesetzt. Datengrundlage sind die offenen Freitextantworten auf Item HOW (vgl. Tabelle 4.5). Jede einzelne Feedbacknachricht bildet eine eigenständige Kodiereinheit. Ziel ist es, die Bandbreite und Charakteristika der von Lehrpersonen formulierten Rückmeldungen systematisch zu erfassen und beschreibbar zu machen. Damit adressiert die Analyse unmittelbar **FF1.2**, die nach den inhaltlichen Ausprägungen und typischen Strategien von Feedback beim Programmierenlernen fragt. Methodische Schritte, die beiden Teilanalysen gemeinsam sind (z. B. Bereinigung, Konsensbildung, Rollen der Kodierenden), wurden bereits in Abschnitt 4.2.4 beschrieben und werden an dieser Stelle nicht erneut ausgeführt.

4.2.5.1 Ausgangspunkt: Kategoriensystem und Erweiterung

Als deduktiver Ausgangspunkt dient das theoretisch fundierte Kategoriensystem von Keuning, Jeuring und Heeren (2019), das Arten von Feedback in automatisierten Lernumgebungen zur Programmierung systematisiert. Die darin enthaltenen Hauptkategorien werden auf das vorliegende Material übertragen und hinsichtlich ihrer Anwendbarkeit auf freiformuliertes, menschliches Feedback von Lehrpersonen geprüft. Subkategorien der Originalsystematik werden zunächst nicht übernommen, da davon auszugehen ist, dass ihre Spezifizität bei offenen Formulierungen zu einer geringeren Trennschärfe führt und eine robuste Zuordnung erschwert.

4.2.5.2 Kodierprozess: Ablauf und Entwicklung des Schemas

Die Entwicklung des Kategoriensystems für die HOW-Analyse erfolgt analog zur WHY-Analyse als regelgeleitetes, iteratives Verfahren (Mayring 2015a):

- **Erste Kodierphase**: Die Sequenzen *Bob* (40 Antworten), *Alice* (33 Antworten) und *Parker* (45 Antworten) werden kodiert, Abweichungen verglichen und in Konsenssitzungen geklärt. Die Kategorien werden durch präzisierte Definitionen und repräsentative Beispiele geschärft.
- **Erweiterung des Schemas**: Auf Basis der überarbeiteten Struktur werden die Sequenzen *Eve* (42 Antworten) und *Dave* (25 Antworten) kodiert. Wo erforderlich, werden deduktive Kategorien inhaltlich angepasst bzw. ergänzt, um wiederkehrende Formvarianten konsistent abzubilden.
- **Abschluss der Kodierung**: Final wird die Sequenz *Jasmine* mit dem stabilisierten Kategoriensystem kodiert.

Inhaltlich werden die Feedbacknachrichten zweidimensional erfasst: (1) *inhaltlicher Fokus* der Rückmeldung (angelehnt an Keuning, Jeuring und Heeren) und (2) *sprachlich-kommunikative Form* (z. B. erklärende Information vs. leitende Frage; vgl. Narciss (2007)). Diese kombinierte Erfassung erlaubt es, Typen und Realisierungsformen von Feedback systematisch zu unterscheiden, ohne den offenen Charakter der Äußerungen einzuschränken. Die Konsensbildung, die Rolle der vier Kodierenden und die Berechnung der Intercoder-Zuverlässigkeit folgen dem in Abschnitt 4.2.4 beschriebenen Verfahren. Tabelle 4.10 dokumentiert die Zuordnung der kodierenden Personen zu den Sequenzen in der HOW-Analyse.

ID	CodingBat		FITech		JetBrains	
	Bob	**Alice**	**Eve**	**Dave**	**Parker**	**Jasmine**
C1	x	x		x		x
C2	x			x	x	x
C3			x		x	
C4		x	x			

Tabelle 4.10: Zuordnung der Kodierenden zu den jeweiligen Datensätzen (HOW-Analyse)

4.3 Entscheidungskriterien für Feedbackinterventionen

Die Analyse der offenen Antworten auf Item WHY führte zur Entwicklung eines Kategoriensystems, das die Entscheidungsgründe von Lehrpersonen für oder gegen die Gabe von Feedback beim Programmierenlernen systematisiert. Damit adressiert die Untersuchung unmittelbar Forschungsfrage **FF1.1** (vgl. Abschnitt 3.2). Zur Überprüfung der Kodierkonsistenz wurde auf Grundlage der Sequenz *Parker* die Intercoder-Reliabilität mit Cohens κ berechnet. Der Wert von 0,68 ist gemäß Landis und Koch (1977) als „substanziell" einzustufen.

4.3.1 Ergebnisse der Auswertung

Tabelle 4.11 fasst das vollständige Kategoriensystem zusammen. Für jede Kategorie werden eine präzise Definition sowie ein Ankerbeispiel aus dem Datenmaterial angegeben, um die empirische Fundierung der Klassifikation nachvollziehbar zu machen.[5] Das resultierende Schema basiert auf insgesamt 464 Begründungen, die im Rahmen der qualitativen Inhaltsanalyse (siehe Abschnitt 4.2.3.3) kodiert wurden. Fünf Kategorien (in Tabelle 4.11 kursiv hervorgehoben) wurden aus der Literatur übernommen (Jeuring et al. 2022) und fanden sich in den Antworten der befragten Lehrpersonen wieder. Die übrigen 14 Kategorien wurden induktiv aus den Daten herausgearbeitet und bilden spezifische, bislang in der Forschung nicht beschriebene Begründungsmuster ab (vgl. Abschnitt 4.2.4.1). Darüber hinaus wurden einzelne der deduktiv übernommenen Kategorien im Kodierprozess inhaltlich verfeinert, um ihre Anwendbarkeit auf die erhobenen Daten sicherzustellen. Diese Anpassungen werden im Folgenden begründet und beschrieben.

Zur besseren Übersicht werden die 19 Kategorien in drei übergeordnete Kategorien eingeordnet, die zugleich eine Rückbindung an Forschungsfrage **FF1.1** ermöglichen:

- **Situative Faktoren**, die sich auf die konkrete Lernsituation beziehen und sowohl prozess- als auch produktbezogene Entscheidungsgründe umfassen,
- **Individuelle Faktoren**, die Zuschreibungen oder Einschätzungen zur lernenden Person betreffen,
- **Sonstige Faktoren**, die keine eindeutige Zuordnung erlauben, in denen jedoch Unsicherheit oder Unentschlossenheit der Lehrperson zum Ausdruck kommt.

[5] Die in Tabelle 4.11 aufgeführten Definitionen und Zitate wurden aus dem Englischen ins Deutsche übertragen. Die Fassung in Originalsprache findet sich in (Lohr et al. 2024a).

In den nachfolgenden Abschnitten werden diese drei Oberkategorien differenziert dargestellt. Entsprechend der Logik der qualitativen Inhaltsanalyse nach Mayring (2015b) erfolgt dabei zunächst die Erläuterung der deduktiv aus der Literatur übernommenen Kategorien, einschließlich ihrer inhaltlichen Präzisierungen im Kodierprozess. Anschließend werden die induktiv entwickelten Kategorien vorgestellt, die neue Entscheidungsgründe erschließen und durch geeignete Ankerbeispiele empirisch veranschaulicht werden. Dieses Vorgehen entspricht dem von Mayring (2015b) beschriebenen Ablauf der Kategorienexplikation, in dem deduktive Vorannahmen mit induktiven Erweiterungen systematisch verbunden werden.

Kategorie	Definition	Ankerbeispiele
DFS *Abweichung von der Aufgabenstellung*	Inkonsistenzen bezüglich der Aufgabenbeschreibung sowie Aufgabenanforderungen, z. B. Änderung der geforderten Signatur der Methode.	„*Bobs Code scheitert daran, mit der Definition der Fibonacci-Funktion konsistent zu sein.*“ „*Die beiden Zeilen sind beide irrelevant für die angegebene Aufgabe.*“ „*Eve beschränkt den Typ des Passworts auf Zahlen, was nicht getan werden sollte.*“
T&E *Trial-and-Error-Verhalten*	Sobald erkennbar wird, dass die Änderungen auf Raten und nicht auf zielgerichtetem Ausprobieren beruhen.	„*Tippen ohne nachzudenken*“ „*Sieht so aus, als ob der/die Lernende Trial-and-Error verwendet, um den Fehler zu finden.*“ „*Willkürliches Programmieren*“
REQUEST *Feedback-Anfrage*	Wenn Lernende um Hilfe bitten (würden).	„*Wenn der/die Lernende um Hilfe gebeten hätte.*“ „*Nur wenn darum gebeten wird.*“
COMPLETE *Absolvierung der Aufgabe*	Wenn Lernende alle Schritte einer Aufgabe absolvieren und eine korrekte Lösung erzielen.	„*Feiere den Erfolg!*“ „*Die Lösung ist korrekt.*“
LOGE *Logischer Fehler*	Programmfehler, die dazu führen, dass das Programm nicht richtig funktioniert.	„*Die Logik ist falsch.*“ „*Es scheint, dass sie die Ziffern sortieren will, aber sie sortiert nur eine Liste der Länge 1.*“
SYNE *Syntax-Error*	Fehler in Bezug auf korrekte Schreibweise, Zeichensetzung, Klammern, Einrückungen etc., die einen Compilerfehler verursachen.	„*Fehlendes Semikolon*“ „*Das letzte `return`-Statement gibt keinen Wert zurück, daher sollte der Compiler diesen Fehler melden.*“
TYPE *Type-Error*	Operationen mit Werten, die nicht dem passenden Datentyp entsprechen, z. B. das Addieren eines Strings und eines Integers.	„*Sie werden eventuell in einen TypeError rennen: `>` wird nicht unterstützt zwischen Instanzen von `str` und `int`*“ „*[Er/Sie] versucht zu überprüfen, ob der String eine Ziffer ist, bevor der Vergleich durchgeführt wird, was trotzdem fehlschlagen wird, wegen des vorherigen TypeErrors.*“
ERROR *Unspezifizierte oder multiple Fehler*	Wenn keine spezifischen Informationen zu dem Fehler vorliegen, auf den sich die Lehrpersonen beziehen, oder wenn mehrere (verschiedene) Fehler vorliegen.	„*Die Fehlermeldung zeigt jetzt einen anderen Typ von Fehler, aber in derselben Zeile.*“ „*Die neue Zeile ist falsch platziert und trägt nicht zur Behebung der anderen Probleme bei.*“ „*Zu viele Probleme demotivieren Anfänger.*“

Fortsetzung auf nächster Seite

Tabelle 4.11 – Fortsetzung von vorheriger Seite

Kategorie	Definition	Ankerbeispiele
STYLE *Stilistische Probleme*	Bestandteile im Code, die zwar keinen Kompilierungsfehler verursachen, aber verbessert werden sollten, z. B. fehlende aussagekräftige Benennungen oder fehlende Kommentare.	„*Die beiden if-Anweisungen können zu einer zusammengefasst werden.*“ „*Es ist besser, für den Then-Teil immer einen Block verwenden.*“ „*Ich würde die beiden Fälle umordnen oder zusammenführen.*“
PERF *Performance-Probleme*	Wenn die Performance oder Laufzeit eines Algorithmus verbessert werden könnte.	„*Ich könnte fragen, ob sie sich eine bessere/schnellere Lösung überlegen kann und das Ganze auf die nächste Ebene bringen (ein iterativer Ansatz statt eines rekursiven).*“ „*Das ist ein guter Moment, um über die Laufzeit des Programms zu sprechen [. . .].*“
START *Start des Problemlösungsprozesses*	Wenn Lernende gerade erst mit dem Problemlösungsprozess begonnen haben, also der erste *Step* angezeigt wird.	„*Ich denke, es ist zu früh für Feedback.*“ „*Sie haben noch nicht genug Code geschrieben.*“
PAUSE *Lange Pause*	Wenn Lernende längere Zeit nichts tun.	„*Nach 4 Minuten ohne Fortschritt ist ein Hinweis nötig, wie es weitergeht.*“
PROGR *Fortschritt*	Den Fortschritt der Lernender ungeachtet kleinerer Verbesserungsmöglichkeiten fördern; den Denkprozess nicht stören; abwarten, ob die Lernenden ihre Fehler selbst erkennen oder ob gute Fortschritte/-deutliche Verbesserungen zu verzeichnen sind.	„*Sie aufzufordern, X zu tun, würde ihren Fortschritt nicht fördern.*“ „*Ich würde abwarten und sehen, wie sie weitermacht.*“ „*Lass sie weitermachen.*“ „*Ich möchte, dass sie das Problem selbst herausfindet.*“ „*Beseitigen von Fehlern, guter Fortschritt.*“
STEPBACK *Verschlechterung* der Lösung	Wenn Lernende sinnvollen Code löschen oder Code bearbeiten, der bereits zuvor korrekt war.	„*Die Änderung hat den Lernenden keiner funktionierenden Implementierung nähergebracht.*“ „*Sie hat ihren sinnvollen Versuch für das return-Statement komplett gelöscht, statt ihn zu überarbeiten.*“
EXFEED *Vorhandenes Feedback*	Wenn es bereits Feedback gibt, das die Lernenden erhalten haben oder erhalten werden, z. B. Fehlermeldungen aus der Lernumgebung, vom Compiler oder von der Lehrkraft.	„*Das Feedback der IDE gibt das passende Feedback.*“ „*Das Tool gibt bereits Feedback*“ „*Bereits bei diesem Meilenstein vermerkt worden und auf das Problem hingewiesen.*“ „*Die Fehlermeldung ist ausreichend.*“
UNC *Unsicherheit*	Wenn die Lehrpersonen unsicher oder unentschlossen sind, warum sie Feedback geben sollen, oder wenn sie nicht verstehen, was Lernende gerade tun.	„*Noch unsicher, was sie mit v2 vorhat.*“ „*Ich habe keine Ahnung, was sie zu tun versucht.*“ „*Ich kann aus der ersten Zeile allein nicht ableiten, was ihr Denkprozess ist.*“
TRUST *Vertrauen in die Lernenden*	Annahmen (und deren Begründung) über die Wahrscheinlichkeit, dass die Lernenden eigenständig Erfolge erzielen (oder nicht), oder über das Ausmaß, in dem die Lernenden Fortschritte erzielen können.	„*Kein Grund anzunehmen, dass sie das nicht hinbekommt.*“ „*Der Lernende sollte das selbst herausfinden können.*“ „*Versteht er das Konzept der Rekursion?*“ „*Ich mache mir Sorgen, dass Parker verwirrt darüber ist, wofür v2 und v3 stehen.*“

Fortsetzung auf nächster Seite

Tabelle 4.11 – Fortsetzung von vorheriger Seite

Kategorie	Definition	Ankerbeispiele
STATEOM *Beurteilung des Geisteszustands der Lernenden*	Wenn explizit ungerechtfertigte Annahmen über den Geisteszustand, die Emotionen, kognitive Prozesse oder die Fähigkeiten der Lernenden getroffen werden.	„*Ich denke, sie ist in Panik.*“, „*Der Lernende ist offiziell verloren.*“ „*Ich muss dieses Programm wahrscheinlich für sie schreiben. Sie ist verloren.*“ „*Sie weichen zu sehr vom Weg ab und werden zu sehr überfordert.*“
PERSONA *Aspekte hinsichtlich der Person*	Expliziter Bezug zu Aspekten, die in der für die Sequenz bereitgestellten Persona erwähnt werden.	„*Bob klingt kompetent genug, und bisher läuft alles gut.*“ „*Das ist ein entscheidender Schritt, da sie mit Rekursion Schwierigkeiten hatte […].*[6]“

Tabelle 4.11: Kategoriensystem zu Feedback-Entscheidungen von Lehrpersonen

4.3.2 Situative Faktoren

Die größte Gruppe der Kategorien verweist auf *situative Merkmale* der Bearbeitung, die für Lehrpersonen als Anlass für oder gegen eine Intervention dienten. Ein Teil der situativen Faktoren geht dabei auf deduktiv übernommene Kategorien aus der Literatur zurück, die im Kodierprozess weiter verfeinert wurden. Ergänzend dazu wurden weitere Kategorien induktiv aus den Begründungen der Teilnehmenden entwickelt. Die situativen Faktoren lassen sich in zwei Subgruppen differenzieren: **produktbezogene** und **prozessbezogene** Entscheidungsgründe, wobei sich erstere auf Eigenschaften des jeweils vorliegenden Artefakts (z. B. Fehlerbilder im Code) beziehen, während letztere die Veränderung dieses Artefakts im Verlauf des Bearbeitungsprozesses betreffen.

4.3.2.1 Produktbezogene situative Faktoren

Die produktbezogenen Faktoren umfassen sowohl deduktiv übernommene als auch im Kodierprozess verfeinerte und induktiv entwickelte Kategorien. Aus der Literatur übernommen wurden **LOGE** (logischer Fehler) sowie **DFS** (Abweichen von der Aufgabenstellung). Die ursprünglich sehr breite Kategorie der *Compiler Errors* wurde differenziert in die drei Unterkategorien **SYNE** (Syntax-Error), **TYPE** (Type-Error) und **ERROR** (unspezifizierte oder multiple Fehler). Ergänzend wurden induktiv neue Kategorien identifiziert, die über die bisherigen Ansätze hinausgehen: **STYLE** (stilistische Probleme), **PERF** (Performance-Probleme) und **EXFEED** (vorhandenes Feedback). Im Folgenden werden die kodierpraktischen Entscheidungskriterien für jede Kategorie erläutert.

SYNE *Kodiert*, wenn formale Sprachverstöße erkennbar sind, die typischerweise sofortige Compiler-/Interpreter-Reaktionen auslösen (z. B. fehlende Klammern, Semikolon, unzulässige Token, fehlerhafte Einrückung in `Python`). *Abgrenzung:* `SYNE` liegt vor, wenn der Parser „abbricht” oder die IDE einen klaren Syntaxmarker setzt. Handelt es sich dagegen um formal korrekten, aber inhaltlich falschen Code, ist `LOGE` einschlägig. Einrückungsfehler wurden aus Gründen der Sprachübergreifbarkeit nicht gesondert geführt, sondern unter `SYNE` subsumiert.

[6] In der Persona-Beschreibung von Alice ist eine Information enthalten, dass sie Rekursion als schwierig empfindet (Siehe Abbildung 4.5)

TYPE *Kodiert*, wenn die Begründungen explizit auf inkompatible Datentypen verweisen oder eine typische Laufzeit-/Typsystemverletzung implizieren (z. B. Vergleich von `str` und `int`, arithmetische Operationen auf Strings). *Abgrenzung:* Während `SYNE` den formalen, *syntaktischen* Bruch markiert, adressiert `TYPE` die *statische/dynamische* Typverträglichkeit. Bei logisch falscher, aber typkorrekter Operation greift hingegen `LOGE`.

ERROR *Kodiert*, wenn Begründungen mehrere unterschiedliche Fehler gleichzeitig adressieren (zu viele Probleme auf einmal) oder die Lokalisierung/Benennung unscharf bleibt (z. B. wechselnde Fehlertypen in derselben Zeile ohne eindeutige Fokussierung). Diese Kategorie dient als Schutzkategorie zur Vermeidung von Scheingenauigkeit. Sie bewahrt die Auswertbarkeit, wenn Antworten bewusst vage bleiben oder eine Häufung von Fehlern im Vordergrund steht.

LOGE *Kodiert*, wenn der Code formal korrekt ist, das beobachtbare Verhalten aber nicht der intendierten Semantik entspricht (z. B. falsches Ergebnis, unvollständige Fallabdeckung, fehlerhafte Rekursionsstruktur). *Abgrenzung:* Keine Syntax-/Typmarker, aber funktionale Inkonsistenz zur Aufgabenlogik.

DFS *Kodiert*, wenn Begründungen explizit auf Regelverletzungen gegenüber der Spezifikation verweisen (z. B. geänderte Signatur, Nichtbeachtung einer geforderten Ausgabeform, Einschränkung des erlaubten Eingabetyps). *Abgrenzung:* Während `LOGE` sich auf die *funktionale Korrektheit* der gewählten Lösung bezieht, markiert `DFS` die Übereinstimmung mit den Anforderungen der Aufgabenstellung.

STYLE *Kodiert*, wenn formale Struktur- und Lesbarkeitsaspekte ohne unmittelbare Funktionsbeeinträchtigung adressiert werden (z. B. redundante Bedingungen, schwer lesbare Verschachtelungen, inkonsistente Benennungen). *Abgrenzung:* Kein `SYNE`/`TYPE`/`LOGE`, sondern Verbesserungsvorschläge zur Wartbarkeit und Klarheit (z. B. „Fälle zusammenführen", „Block konsequent setzen").

PERF *Kodiert*, wenn Effizienzfragen als Anlass genannt werden (z. B. Vorschlag eines iterativen gegenüber einem rekursiven Ansatz, Reduktion unnötiger Durchläufe). *Abgrenzung:* Funktional korrekter Code wird aus didaktischen Gründen auf Laufzeit-/Ressourcenaspekte reflektiert.

EXFEED *Kodiert*, wenn die Entscheidung einer Intervention explizit mit bereits vorhandenen Rückmeldungen begründet wird (IDE-/Compiler-Hinweise, frühere Annotationen).

In der Summe zeigen diese Kategorien, dass situative Faktoren bei Feedback-Entscheidungen nicht nur an *Fehlern* im engen Sinn festgemacht werden, sondern auch Spezifikationskonformität (`DFS`), Codequalität (`STYLE`), Effizienzüberlegungen (`PERF`) und vorhandene Rückmeldungen (`EXFEED`) systematisch einbeziehen.

4.3.2.2 Prozessbezogene situative Faktoren

Ein weiterer Teil der Antworten bezog sich nicht auf den Programmcode selbst, sondern auf die zeitliche Dynamik und den beobachtbaren Verlauf der Bearbeitung. Diese Rückmeldungen thematisieren Entscheidungssituationen, die sich aus dem Prozesscharakter von Lernaktivitäten ergeben. Die resultierenden Kategorien zu prozessbezogenen Entscheidungskriterien enthalten ebenfalls sowohl deduktiv übernommene als auch im Kodierprozess verfeinerte und induktiv entwickelte Kategorien. Deduktiv herangezogen wurden **T&E** (Trial-and-Error-Verhalten) und

Subgoal Completion (Jeuring et al. 2022). **T&E** wurde weitgehend übernommen und präzisiert. Die Kategorie *Subgoal Completion* konnte mangels explizit formulierter Teilziele in den vorliegenden Aufgabenstellungen des Materials nicht direkt übernommen werden und wurde deshalb regelgeleitet unter **PROGR** (Fortschritt) subsumiert. Zusätzlich wurden mehrere Kategorien induktiv entwickelt. Ergänzt wurden die Kategorien **START**, **COMPLETE**, **STEPBACK**, **TRUST**, **PAUSE** sowie **REQUEST**. Im Folgenden werden die kodierpraktischen Kriterien und Abgrenzungen dargestellt.

T&E *Kodiert*, wenn in den Begründungen explizit auf ungerichtetes, zufälliges Ausprobieren verwiesen wurde (z. B. Aussagen, dass Änderungen ohne Nachdenken oder planlos vorgenommen wurden). *Abgrenzung:* Wurde in den Rückmeldungen dagegen betont, dass ein systematisches oder hypothesengeleitetes Vorgehen erkennbar sei, wurde die Begründung nicht unter `T&E`, sondern unter `PROGR` (Fortschritt) eingeordnet.

PROGR *Kodiert*, wenn Lehrpersonen als Begründung auf erkennbaren Fortschritt im Bearbeitungsprozess verwiesen, etwa durch die schrittweise Beseitigung von Fehlern oder durch sichtbare Verbesserungen. *Abgrenzung:* Begründungen, die lediglich planloses oder erratisches Verhalten beschrieben, wurden nicht `PROGR`, sondern `T&E` zugeordnet.

TRUST *Kodiert*, wenn Lehrpersonen ihre Entscheidung darauf bezogen, inwiefern Lernende den nächsten Bearbeitungsschritt eigenständig bewältigen können. Dieses Vertrauen konnte sich entweder auf den bisherigen Prozessverlauf stützen (z. B. erkennbare Selbstkorrekturen, kontinuierlicher Fortschritt) oder auf Merkmale der Persona wie Fähigkeiten, Vorerfahrungen oder bekannte Schwierigkeiten. *Abgrenzung:* Wurde ausschließlich auf eine aktuelle Verbesserung des Codes, ohne Bezug zu Eigenschaften der Lernenden oder dem bisherigen Prozess verwiesen, so erfolgte die Kodierung unter `PROGR`.

START *Kodiert*, wenn Begründungen ausdrücklich auf den allerersten Bearbeitungsschritt Bezug nahmen. Z. B. wurde hier angeführt, dass es zu früh für Feedback sei, da noch nicht genug Bearbeitungsstand vorlag, um eine fundierte Rückmeldung zu rechtfertigen. *Abgrenzung:* Begründungen, die auf frühe, aber bereits substanzielle Fortschritte verwiesen, wurden unter `PROGR` und nicht unter `START` eingeordnet.

COMPLETE *Kodiert*, wenn Lehrpersonen den erfolgreichen Abschluss der Aufgabe als Anlass für Feedback nannten, meist in Form von Bestätigung oder Lob. *Abgrenzung:* Wurde nicht der Abschluss, sondern der Weg dorthin (z. B. ein Teilerfolg) als Grund benannt, wurde die Begründung der Kategorie `PROGR` zugeordnet.

STEPBACK *Kodiert*, wenn Begründungen explizit auf eine Verschlechterung des Lösungsstandes verwiesen (z. B. das Löschen von korrektem Code). *Abgrenzung:* Begründungen, die sich auf konzeptuelle Fehler ohne explizite Verschlechterung des bestehenden Codes bezogen, wurden mittels produktbezogener Kategorien zugeordnet.

PAUSE *Kodiert*, wenn längere Inaktivität als Anlass für Feedback genannt wurde.

REQUEST *Kodiert*, wenn Lehrpersonen ihre Entscheidung damit begründeten, dass Feedback nur auf explizite oder hypothetische Anfragen der Lernenden erfolgen solle. *Abgrenzung:* Wurde die Entscheidung nicht an die Initiative der Lernenden, sondern an situative Merkmale wie Fortschritt oder Pausen geknüpft, wurde die Begründung den entsprechenden Kategorien zugeordnet.

4.3.2.3 Individuelle Faktoren

Neben situativen Kriterien spielten in den Begründungen auch individuelle Faktoren eine Rolle. Diese beziehen sich nicht auf die Artefakte oder den Bearbeitungsprozess selbst, sondern auf wahrgenommene Eigenschaften, Fähigkeiten oder angenommene geistige Zustände der Lernenden. Sie spiegeln damit persönliche Annahmen und subjektive Deutungen wider, die die Entscheidung für oder gegen eine Intervention beeinflussten. Alle Kategorien zu den individuellen Faktoren entstanden induktiv aus dem Datenmaterial.

PERSONA *Kodiert*, wenn Lehrpersonen ihre Entscheidung explizit mit Informationen aus der bereitgestellten Persona begründeten, etwa zu Motivation, Vorerfahrung oder biografischen Merkmalen. Diese Kategorie zeigt, dass kontextualisierte Zusatzinformationen in den Feedbackentscheidungen berücksichtigt wurden. *Abgrenzung:* Wenn die Begründung nicht auf Merkmale der Persona, sondern beobachtbares Verhalten im Code oder Prozess verwies, wurde die Kodierung den entsprechenden situativen Kategorien zugeordnet.

STATEOM *Kodiert*, wenn Lehrpersonen ihre Entscheidung mit angenommenen emotionalen oder kognitiven Zuständen der Lernenden begründeten, etwa Unsicherheit, Überforderung oder weiterer Zustände wie „Verlorensein". Diese Zuschreibungen waren häufig vage – ohne Bezug zu konkreten Code- oder Prozessmerkmalen.

UNC *Kodiert*, wenn Lehrpersonen ihre eigene Unsicherheit hinsichtlich der Einschätzung artikulierten, etwa weil die Intentionen der Lernenden unklar blieben oder der nächste Bearbeitungsschritt nicht nachvollzogen werden konnte. *Abgrenzung:* Begründungen, die nicht die Unklarheit der Lehrperson, sondern die der Lernenden thematisierten, wurden `STATEOM` oder anderen passenden Kategorien zugeordnet.

Nicht kodierbare und ausgenommene Antworten

Neben den inhaltlich relevanten Kategorien wurden drei zusätzliche Labels vergeben, um Antworten systematisch zu kennzeichnen, die nicht sinnvoll kodierbar sind und deshalb von der Analyse ausgeschlossen werden. Diese Labels sind nicht im Kategoriensystem in Tabelle 4.11 enthalten, da sie keine Rückschlüsse auf die eigentlichen Feedbackentscheidungen zulassen und somit **FF1.1** nicht adressieren. Im Zuge der methodischen Transparenz werden diese aber dennoch hier aufgeführt.

Mit dem Label **IGNORE** wurden allgemeine Kommentare oder Bemerkungen ohne Bezug zur gestellten Frage markiert – etwa persönliche Einschätzungen zur Umfrage oder Hinweise an das Forschungsteam. Mit **CONF** (*confusion*) wurden Antworten kodiert, bei denen erkennbar ist, dass Teilnehmende die Systematik der Umfrage missverstanden haben. Dies betrifft insbesondere Fälle, in denen die gegebene Rückmeldung den in den Instruktionen definierten Rahmenbedingungen widerspricht und daher nicht gültig ist. Ein typisches Beispiel ist die fälschliche Annahme, dass zuvor eingegebenes Feedback von den Lernenden berücksichtigt werde, obwohl es sich um fest vorgegebene Bearbeitungssequenzen handelt, in denen das Feedback keine Wirkung entfalten kann. Das Label **NOR** (*no reason*) wurde verwendet, wenn keine inhaltlich nachvollziehbare Begründung vorliegt, etwa bei leeren Antworten (vgl. Datenbereinigung in Abschnitt 4.2.3.1).

Während `NOR`-Fälle in der Regel bereits im Bereinigungsschritt eindeutig zu identifizieren waren, zeigt sich bei Antworten die `CONF` bzw. `IGNORE` gelabeld wurden, dies häufig erst bei der genaueren Analyse im Rahmen des Kodierprozesses, dass die jeweilige Begründung nicht inhaltlich auswertbar ist.

4.4 Charakteristika gegebener Feedbacknachrichten

Der zweite Teil der Analyse richtet den Fokus auf die *inhaltliche Ausgestaltung* der von Lehrpersonen formulierten Feedbacknachrichten. Damit wird unmittelbar Forschungsfrage **FF1.2** adressiert, die untersucht, welche inhaltlichen Ausprägungen die von Lehrpersonen formulierten Rückmeldungen im Rahmen von Programmierprozessen haben können (vgl. Abschnitt 3.2).

Zur systematischen Erfassung der Rückmeldungen wurde das von Keuning, Jeuring und Heeren (2019) entwickelte Kategoriensystem herangezogen, das verschiedene Arten automatisierten Feedbacks in Lernumgebungen klassifiziert. Diese Taxonomie diente als deduktiver Ausgangspunkt für die Kodierung. Im Kodierprozess wurde das System iterativ geprüft, an die Charakteristika menschlicher Rückmeldungen angepasst und durch induktiv entwickelte Kategorien erweitert, um empirisch beobachtete Formen von Feedback durch Lehrpersonen adäquat abzubilden.

4.4.1 Ergebnisse der Auswertung

Das resultierende Kategoriensystem ist in Tabelle 4.12 dargestellt. Wie bereits bei der Darstellung der Ergebnisse der `WHY`-Analyse wird für jede Kategorie eine präzise Definition sowie ein Ankerbeispiel aus dem Datenmaterial angegeben, um die empirische Fundierung der Klassifikation nachvollziehbar zu machen.[7]

Zur besseren Übersicht werden die Kategorien inhaltlich differenziert dargestellt. Entsprechend der Logik der qualitativen Inhaltsanalyse nach Mayring (2015b) werden dabei zunächst die deduktiv übernommenen Kategorien erläutert, einschließlich der im Kodierprozess vorgenommenen Präzisierungen. Anschließend folgen die induktiv entwickelten Kategorien, die neue Facetten der Feedbackgestaltung erschließen und anhand von Ankerbeispielen empirisch illustriert werden. Dieses Vorgehen entspricht der von Mayring (2015b) beschriebenen Kategorienexplikation, bei der deduktive Vorannahmen systematisch mit induktiv gewonnenen Erkenntnissen verbunden werden.

4.4.1.1 Inhaltliche Ausgestaltung: Deduktive Kategorien

Aufgabenbezogenes Feedback (KTC). Diese Hauptkategorie umfasst Rückmeldungen, die direkt an die konkrete Aufgabenstellung gebunden sind. Kodiert wurde hier insbesondere die aus der Literatur übernommene Subkategorie `KTC-TPR`, die grundlegende prozedurale Hinweise umfasst. Dazu zählen etwa Rückmeldungen zum Debugging (*„Kommentiere die fehlerhafte Zeile zunächst aus.“*) oder Aufforderungen, den Programmablauf manuell nachzuvollziehen (*„Geh den Code einmal auf Papier Schritt für Schritt durch.“*). Solche Rückmeldungen adressieren keine inhaltlichen Konzepte, sondern unterstützen allgemeine Strategien der Fehleranalyse und Problemlösung im Rahmen der Aufgabenstellung. Ergänzend dazu wurde die Subkategorie `KTC-TR` verwendet, die sich auf Rückmeldungen zu formalen Aufgabenvorgaben bezieht. Kodiert wurden hier etwa Verweise auf einzuhaltende Konventionen oder Wiederholungen bereits erfüllter Anforderungen, die den Fortschritt würdigen und die Angemessenheit des Vorgehens verdeutlichen (*„Du hast erfolgreich eine Wiederholung implementiert und vergleichst jedes Zeichen mit dem bisherigen Maximum.“*).

Konzeptuelles Feedback (KC). Kodiert wurden hier Rückmeldungen, die über den unmittelbaren Aufgabenkontext hinausgehen und abstrakte Programmierkonzepte betreffen. Unter `KC-EXP` wurden Erklärungen und Definitionen subsumiert, etwa zu rekursiven Strukturen oder

[7] Die in Tabelle 4.12 aufgeführten Definitionen und Zitate wurden aus dem Englischen ins Deutsche übertragen. Die Fassung in Originalsprache findet sich in (Lohr et al. 2025a).

Kontrollflusskonstrukten. Nur eine einzelne Rückmeldung ließ sich eindeutig der Subkategorie `KC-EXA` (konzeptionelle Beispiele) zuordnen. Abgegrenzt wurden Fälle, in denen der konzeptuelle Gehalt zwar erkennbar war, sich jedoch eng auf die jeweilige Aufgabenstellung bezog bzw. auf Konzepte einer anderen Fachdisziplin (z. B. Fibonacci-Formel) – diese wurden regelgeleitet unter KTC-TR und nicht unter KC eingeordnet.

Fehlerbezogenes Feedback (KM). Diesee Hauptkategorie umfasst Rückmeldungen, die sich explizit auf Fehlerbilder im Code beziehen. Mit `KM-CE` wurden Verweise auf Syntaxfehler und durch Compiler/IDE markierte Probleme kodiert, einschließlich Tipp- und Einrückungsfehlern. Die Kategorie `KM-TF` wurde im Kodierprozess erweitert und umfasst nicht nur Hinweise auf fehlgeschlagene Tests, sondern auch positive Rückmeldungen zu bestandenen Testfällen. Damit wurde die ursprüngliche Definition aus der Literatur systematisch angepasst.

Prozedurales Vorgehensfeedback (KH). Kodiert wurde hier Feedback, das auf das weitere Vorgehen im Problemlöseprozess abzielt. Unter der Kategorie `KH-EC` wurden Hinweise zur konkreten Fehlerbehebung erfasst (*„Füge eine Bedingung ein, um `NULL`-Werte abzufangen.“*). Die Subkategorie `KH-TPS` umfasst strategische Anleitungen zum strukturierten Weiterarbeiten, etwa durch Zwischenschritte. Hinweise auf Stil- oder Effizienzverbesserungen wurden unter `KH-IM` kodiert. Diese traten vor allem bei bereits semantisch korrekten Programmen auf, wurden jedoch vereinzelt auch in früheren Bearbeitungsphasen gegeben.

4.4.1.2 Inhaltliche Ausgestaltung: Induktive Kategorien

Im Kodierprozess zeigte sich, dass nicht alle Rückmeldungen durch die deduktiv übernommenen Kategorien nach Keuning, Jeuring und Heeren (2019) erfasst werden konnten. Bestimmte Formen von Feedback wiesen Merkmale auf, die weder inhaltlich noch funktional eindeutig den bestehenden Kategorien zugeordnet werden konnten. Um diese empirisch beobachteten Phänomene angemessen abzubilden und die Vollständigkeit des Kategoriensystems sicherzustellen, wurden zwei zusätzliche Kategorien induktiv entwickelt:

Motivationales Feedback (MOT). Diese Kategorie umfasst Rückmeldungen, die nicht auf fachliche Inhalte zielen, sondern der positiven Verstärkung und Ermutigung dienen. Beispielsweise wurde Lernenden zu gelungenen Bearbeitungsschritten gratuliert oder ihre Fortschritte anerkennend kommentiert. Eine Zuordnung zur Kategorie `MOT` erfolgte jedoch nur dann, wenn entweder eine explizite motivationale Intention geäußert wurde (“Motiviere den Schüler (auf dem richtigen Weg mit +)”) oder die Formulierung eindeutig motivationsfördernden Charakter hatte und an die Lernenden gerichtet war (“Super! Gut gemacht!”).

Informationsanfrage (GETI). Rückmeldungen, die primär der Einholung zusätzlicher Informationen dienen, um z. B. weitere Informationen über den Denkprozess der Lernenden zu erhalten (*„Was hast du dir bei dieser Zeile gedacht?“*) wurden mit `GETI` kodiert.

4.4.1.3 Format des Feedbacks: Information oder Leitfrage

Neben den inhaltlichen Aspekten wurde im Kodierprozess auch das kommunikative Format der Rückmeldungen berücksichtigt. Bereits der ersten Probekodierung der `HOW`-Items zeigte sich, dass sich die Rückmeldungen der Lehrpersonen nicht nur hinsichtlich ihrer inhaltlichen Ausgestaltung unterscheiden, sondern auch durch ihre sprachlich-kommunikative Form. Dieses Merkmal ließ sich nicht allein über die inhaltlichen Kategorien abbilden, sondern erforderte eine ergänzende Dimension.

Zwei zentrale Ausprägungen wurden identifiziert: Einige Rückmeldungen enthielten *direkte Informationen*, etwa durch explizite Hinweise auf Fehler im Code ("Es fehlt ein Semikolon am Ende der Zeile 5"). Andere formulierten hingegen *offene Leitfragen*, die Lernende zu einer eigenständigen Reflexion über ihren Code oder zugrunde liegende Konzepte anregen sollten ("Was wäre ein guter Variablenname?"). Im Sinne der qualitativen Inhaltsanalyse nach Mayring (2015a) wurde diese Beobachtung in das Kategoriensystem integriert und als zweite, orthogonal zur inhaltlichen Klassifikation stehende Dimension systematisch kodiert. Dabei wurden zwei Kategorien unterschieden:

INFO (ⓘ). *Kodiert*, wenn die Rückmeldungen direkt Informationen vermitteln, beispielsweise durch konkrete Hinweise auf Fehler oder durch explizite Erklärungen zu Vorgehensweisen im Code. Diese Form des Feedbacks stellt den Lernenden unmittelbar verwertbares Wissen bereit, ohne dass weitere eigene Reflexion erforderlich ist. Wichtig ist, dass auch in Frageform formulierte Aussagen dieser Kategorie zugeordnet werden können, sofern sie inhaltlich eindeutige Informationen bereitstellen. Ein Beispiel ist die Rückmeldung: "Solltest du vielleicht ein anderes Argument verwenden?". Trotz Frageform wird hier unmittelbar eine konkrete Korrekturmöglichkeit benannt und somit Information vermittelt.

REFLECT (Ⓡ). *Kodiert*, wenn Rückmeldungen in Form von reflektierenden oder leitenden Fragen gegeben werden, die Lernende dazu anregen, ihre eigenen Lösungsansätze kritisch zu hinterfragen oder zugrunde liegende Konzepte selbstständig zu erschließen. Entscheidend ist auch hierbei die intendierte Aktivierung eigenständiger Denkprozesse, nicht die grammatikalische Formulierung als Frage. So können auch inhaltlich offene Aussagen dieser Kategorie zugeordnet werden, sofern sie eine Reflexion fördern (*„Versuch zu überlegen, was die Fehlermeldung versucht dir zu sagen.“*).

Die Ergänzung dieser Dimension folgt zwei Zielen: Erstens ermöglicht sie eine präzisere Erfassung der Kommunikationsformen, die Lehrpersonen in realen Feedbacksituationen wählen. Zweitens trägt sie dazu bei, Unterschiede zu typischem automatisiertem Feedback in Lernumgebungen sichtbar zu machen, das überwiegend auf informativen Hinweisen beruht. In den Ankerbeispielen von Tabelle 4.12 sind die beiden Kategorien dieser zweiten Dimension jeweils eindeutig markiert.

Kat.	Definition	Ankerbeispiele mit Format ⓘ oder Ⓡ
KTC	**Knowledge about Task Constraints**	
TR	Fokussiert auf konkrete *Aufgabenanforderungen* an eine korrekte Löunsg. Umfasst u. a. geforderte Paradigmen (z. B. Einsatz von Rekursion), Benennungskonventionen oder auferlegte Beschränkungen (z. B. keine Java-API).	Ⓡ *„Lass uns die Aufgabenstellung noch einmal lesen, um sicherzugehen, dass wir sie korrekt verstanden haben.“* ⓘ *„Ich werde kurz versuchen die Formel der Fibonacci-Folge zu erklären.“*
TPR	Hinweise zu *Regeln der Aufgabenbearbeitung* stellen allgemeine Informationen zur Herangehensweise an die Aufgabe bereit. Dazu gehören auch allgemeine Erläuterungen zur Lösung bestimmter Aufgabentypen (z. B. wie man bei rekursiver Programmierung grundsätzlich vorgeht). Außerdem enhält es Hinweise zum prozeduralen Wissen in der Programmierung, wie z. B. Debugging-Strategien oder Code Tracing	Ⓡ *„Ich würde ihn fragen, welchen Wert n haben müsste, um den rekursiven Aufruf zu erreichen.“* ⓘ *„Wenn du eine rote Linie siehst, bedeutet es, dass ein Syntaxfehler vorliegt.“* ⓘ *„Versuch es noch einmal auszuführen. Was macht es jetzt?“*
KC	**Knowledge about Concepts**	

Fortsetzung auf nächster Seite

Tabelle 4.12 – Fortsetzung von vorheriger Seite

Kat.	Definition	Ankerbeispiele mit Format ⓘ oder Ⓡ
EXP	*Erläuterungen zum Thema*: Erläuterungen oder Hinweise zur Wiederholung bestimmter Programmierkonzepte, die im Zusammenhang mit der Programmierung relevant sind. Andere Konzepte, wie beispielsweise mathematische Formeln, die für die Aufgabe erforderlich sind, fallen in die Kategorie KTC-TR.	ⓘ „*Zusätzliche Erläuterungen zu Attributen von Objekten/Zeichenfolgen*“ ⓘ „*Erläuterung zur Funktionsweise von isdigit() und weiteren hilfreichen Informationen für die Aufgabe.*“ ⓘ „*Die Struktur einer bedingten Anweisung lautet: if(Bedingung){Anweisung}*“ Ⓡ „*[Ich würde] erklären, wie man Typumwandlungen in Python durchführt.*“
EXA	Beispiele, die Programmierkonzepte veranschaulichen (z. B. Analogien) wie Verweise auf Kursmaterial, das Beispiele für das Konzept enthält.	ⓘ „*Ich würde sie an den Zweck von Semikolons erinnern (mit der Analogie, dass Semikolons wie Punkte in Sätzen sind).*“ Ⓡ „*Überprüfe das Kursmaterial, um zu sehen, wie If-Anweisungen aussehen sollten.*“
KM	**Knowledge about Mistakes**	
TF	Verweise auf *Testergebnisse* (z. B. bereitgestellte Unit-Tests).	Ⓡ „*Schau dir den ersten Fall an, der hier fehlschlägt. Warum ist das so?*“ Ⓡ „*Warum denkst du, dass Fall 0 und 1 bestehen, aber die restlichen fehlschlagen?*“
CE	Verweis auf *Syntaxfehler* im Code (z. B. Einrückungsfehler, fehlende Klammer/Semikolon, Typfehler), die in der Regel von einem Compiler oder Interpreter erkannt werden.	Ⓡ „*Versuch zu überlegen, was die Fehlermeldung versucht dir zu sagen.*“ Ⓡ „*Denk über die Werte von v3 und v2 nach. Können sie verglichen werden? Repräsentieren sie den gleichen Datenwert?*“ ⓘ „*Beachte, dass v1 ein str ist, v2 jedoch ein int.[…] Python ist verwirrt, weil du eine Zahl (0) mit einem Zeichen (einer Ziffer) vergleichst.*“
SE	Verweis auf *Semantische Fehler* in der Lösung (z. B. Logikfehler, falsche Verwendung von Operatoren)	Ⓡ „*Was möchtest du mit v3.isdigit erreichen? Warum brauchst du so etwas?*“ Ⓡ „*Ist es (wirklich) notwendig, dass du deine Variable in einen int parst?*“
SI	Verweis auf *Stilistische Probleme*, z. B. wenn der Code unleserlich ist, bestimmte Konventionen bei der Benennung von Variablen nicht eingehalten werden oder wenn der Code schlecht formatiert ist.	ⓘ „*Es ist besser, für den then-Teil immer einen Block einzuführen.*“ Ⓡ „*Er […] sollte es sich nun als Ganzes ansehen und überlegen, ob er irgendwelche Verbesserungen am Design oder Stil vornehmen möchte.*“ Ⓡ „*Ich würde Parker fragen wofür v2 und v3 steht und ob das gute Variablennamen sind.*“
PI	Verweis auf *Performance-Probleme*, z. B. wenn das Programm eine schlechte Laufzeit hat oder unnötig viel Speicherplatz benötigt.	Ⓡ „*Ich könnte sie fragen, ob ihr eine bessere/schneller Lösung einfällt und [den Code] so auf die nächste Stufe bringen.*“ Ⓡ „*Was [ist] die Laufzeit des Programmes in Bezug auf n? Gibt es irgendeine Möglichkeit, die Aufgabe mit einem schnelleren Programm zu lösen?*“
KH	**Knowledge about how to proceed**	
EC	Hinweise zur Fehlerbehebung bei „Bugs”: Konzentriert sich darauf, wie ein bestimmter Syntax-/Semantikfehler im Programm behoben werden kann.	Ⓡ „*Ich würde bitten, mir den Lösungsprozess Schritt für Schritt zu erklären.*“ ⓘ „*Alles sollte in derselben Spalte angeordnet sein, anstatt in if/else/while-Blöcken.*“ ⓘ „*Ich würde darauf hinweisen, dass Zeile 6 v3 = int(v3) lauten muss.*“

Fortsetzung auf nächster Seite

Tabelle 4.12 – Fortsetzung von vorheriger Seite

Kat.	Definition	Ankerbeispiele mit Format ⓘ oder Ⓡ
TPS	Rückmeldungen zu Schritten der Aufgabenbearbeitung: Gibt Hinweise bzw. regt zum Nachdenken darüber an, wie man vorgehen muss, um bei der Bearbeitung einen Schritt weiterzukommen.	Ⓡ „*Vervollständige den Code und ignoriere die Fehler erstmal.*“ ⓘ „*Dieses print statement könnte dir helfen. Vielleicht kannst du ein anderes Argument benutzen??*“
IM	Konzentriert sich darauf, wie die Qualität des Programms (Stil, Performance) verbessert werden kann..	Ⓡ „*Kannst du diese beiden if-Statements nicht kombinieren?*“ ⓘ „*Du könntest auch eine einzige Bedingung formulieren: if (n == 0 || n ==1) { return n }*“ ⓘ „*Variablennamen helfen oft dabei, den Typ einer Variablen zu verstehen: input_ string, one_ char, highest_ char*“
	Weitere Kategorien	
KMC	Rückmeldungen zu Strategien, die zu verwenden sind, um das Problem zu lösen. Beinhaltet auch das Anbieten von Hilfe bei Schwierigkeiten; nicht im Zusammenhang mit bestimmten Fehlern zu verwenden.	ⓘ „*Überlege erst, bevor du tippst.*“ Ⓡ „*Ich würde fragen, ob sie Fragen zum weiteren Vorgehen haben.*“
GETI	Rückmeldungen, die der Einholung zusätzlicher Informationen über Gedanken/Absichten/Wissen der Lernenden dienen.	Ⓡ „*Es ist nicht klar, worauf sich ‘My‘ hier bezieht. Meinst du eine Variable ‘My‘ oder möchtest du das Wort ‘My‘ auf dem Bildschirm ausgeben?*“ Ⓡ „*Keine Ahnung, zuerst möchte ich herausfinden, was sie über dieses Programm denkt.*“
MOT	Rückmeldungen, die darauf abzielen, Lernende zu motivieren.	ⓘ „*Den Lernenden motivieren (auf dem richtigen Weg mit +).*“ ⓘ „*Ich würde zum Bestehen aller Tests gratulieren und sagen, dass eine gute Lösung gefunden wurde.*“

Tabelle 4.12: Kategoriensystem zu Feedback-Entscheidungen von Lehrpersonen

4.5 Diskussion der Ergebnisse

Die Ergebnisse der `WHY`-Analyse zeigen ein konsistentes und zugleich differenziertes Bild didaktischer Entscheidungspraxis. Die entwickelten Kategorien erweitern deduktiv übernommene Ereignistypen (z. B. `Compiler-/Syntaxfehler`, `semantische Fehler`, `Trial & Error`) um prozess- und kontextsensitive Aspekte (`PROGR`, `PAUSE`, `STEPBACK`, `TRUST`, `REQUEST`, `EXFEED`) sowie qualitäts- und effizienzbezogene Kriterien (`STYLE`, `PERF`, `DFS`). Damit wird sichtbar, dass Eingriffsentscheidungen in der Lehrpraxis nicht allein an „Fehlern” festgemacht werden, sondern systematisch auch Spezifikationskonformität, Codequalität, Laufzeitüberlegungen und bereits vorliegende Rückmeldungen einbeziehen. Dies stützt die Befundlage, dass Feedbackentscheidungen sowohl produkt- als auch prozessorientierte Signale integrieren.

Ein Abgleich mit dem etablierten Feedbackmodell von Hattie und Timperley (2007) zeigt, dass sich die identifizierten Entscheidungskategorien den vier Feedback-Ebenen systematisch zuordnen lassen: Kategorien wie `DFS`, `LOGE`, `SYNE`/`TYPEE`, `ERROR`, `STYLE`, `PERF` und `STEPBACK` adressieren die *Aufgabenebene*, indem sie auf spezifische Anforderungen oder konkrete Probleme im Code fokussieren. `START`, `T&E`, `PAUSE`, `PROGR` und `UNC` verweisen auf die *Prozess-Ebene*, da sie Bearbeitungsverläufe, Unsicherheit und Strategiewechsel markieren. Die *Selbstregulationsebene* wird insbesondere in `REQUEST`, `TRUST` und `STATEOM` sichtbar, die Einschätzungen zur Eigenständigkeit, Motivation oder emotionalen Lage der Lernenden einbeziehen. `PERSONA` verweist auf die *Selbst-Ebene* durch Annahmen über individuelle Merkmale der Lernenden. Diese Passung verdeutlicht,

dass die vorliegenden Kategorien ein in der Literatur breit rezipiertes Wirkmodell von Feedback differenziert im Programmierkontext operationalisieren.

Für die Interpretation lassen sich zwei Aspekte hervorheben. Erstens traten *gleichartige Begründungen* (z. B. erkennbarer Fortschritt oder vorhandenes IDE-Feedback) teils als Anlass *für*, teils *gegen* ein Eingreifen auf. Dies deutet auf eine situationsabhängige *Abwägungslogik* in der Entscheidungspraxis hin: Lehrpersonen wägen jeweils ab, ob ein Eingriff den Lernprozess unterstützt oder stört, und berücksichtigen dabei die wahrgenommene Selbstständigkeit und Kompetenz der Lernenden (z. B. `TRUST`). Zweitens zeigen die Analysen, dass die Entscheidungen der Lehrpersonen häufig zeitliche und prozessuale Aspekte der Bearbeitung berücksichtigen, etwa den Beginn (`START`), Unterbrechungen (`PAUSE`), Rückschritte (`STEPBACK`) oder Fortschrittsverläufe (`PROGR`). Damit rückt die Betrachtung stärker von einzelnen Fehlerereignissen hin zu *prozessbezogenen Indikatoren*, die für formative Feedbackentscheidungen von besonderer Bedeutung sind.

Für die Konzeption adaptiver Systeme ergeben sich drei Implikationen. Erstens braucht es *beobachtbare* und *objektivierbare* Auslöser für prozessbezogene Kategorien (z. B. `PAUSE`, `STEPBACK`, `PROGR`), um Entscheidungsregeln maschinenlesbar zu gestalten. Zweitens sollten *nicht direkt beobachtbare*, interpretativ erschlossene Kategorien (`TRUST`, `STATEOM`, `PERSONA`) zurückhaltend und nur in Verbindung mit robusten Verlaufssignalen genutzt werden, um Fehlschlüsse über individuelle Dispositionen zu vermeiden. Drittens legt `EXFEED` nahe, Feedbackquellen zu *orchestrieren* statt zu duplizieren, d. h. automatisches, IDE- und Lehrendenfeedback sind in einer kohärenten Priorisierung zusammenzuführen. Diese Ableitungen korrespondieren mit der Zielsetzung, Entscheidungsstrukturen als Grundlage für kontextsensitive Feedbackprozesse zu formalisieren (Kap. 4.3).

Abschließend ist zu betonen, dass die `WHY`-Ergebnisse *deskriptiv* zu verstehen sind: Sie beschreiben professionelle Entscheidungsmuster, ohne deren *Wirksamkeit* zu normieren. Eine normative Bewertung — etwa im Sinne „richtiger“ Eingriffsschwellen — erfordert experimental- oder feldbasierte Wirksamkeitsprüfungen, idealerweise mit prozessorientierter Datenerhebung und Variation der Interventionslogik.

Die HOW-Analyse adressiert **FF1.2** und untersucht, wie Lehrpersonen inhaltlich und sprachlich auf die von ihnen wahrgenommenen Situationen reagieren. Ausgangspunkt war die Übertragung der Taxonomie von Keuning, Jeuring und Heeren (2019), die auf der Analyse von automatisiertem Feedback in Lernumgebungen basiert. Im Vergleich dazu zeigt das Feedback der befragten Lehrpersonen eine deutlich größere Bandbreite sowohl in der inhaltlichen Tiefe als auch in der sprachlichen Ausgestaltung. Besonders auffällig war der häufige Einsatz offener Leitfragen, die weniger auf die direkte Bereitstellung von Informationen als vielmehr auf die Aktivierung und Selbstreflexion der Lernenden abzielen (z. B. `REFLECT/INFO`). Diese Erweiterung der Taxonomie um eine kommunikative Dimension erlaubt es, nicht nur den inhaltlichen Fokus, sondern auch die Interaktionsform systematisch zu erfassen. Damit wird sichtbar, dass Lehrende Feedback als dialogischen Prozess verstehen, der Lernende zum Nachdenken und zur Selbstkorrektur anregen soll – im Gegensatz zu automatisierten Rückmeldungen, die primär auf Fehlerkorrektur ausgerichtet sind.

Zahlreiche Rückmeldungen umfassten mehrere Aspekte gleichzeitig, etwa motivierende Elemente, prozedurale Hinweise oder konkrete Verbesserungsvorschläge. Diese Mehrschichtigkeit verdeutlicht, dass menschliches Feedback häufig mehrere Funktionen erfüllt – etwa das gleichzeitige Unterstützen, Fordern und Bestätigen. So kann eine Äußerung wie “Start with the error message” je nach Kontext verschiedene Intentionen verfolgen: Sie kann als Information über eine Fehlerquelle (`KM-CE-`ⓘ), als Anregung zur Reflexion (`KM-CE-`Ⓡ) oder als strategischer Hinweis zum weiteren Vorgehen (`KTC-TPR-`ⓘ) interpretiert werden. Damit wird deutlich, dass die kommunikative Form – Frage, Hinweis, Erklärung – nicht eindeutig Rückschlüsse auf die zugrundeliegende pädagogi-

sche Absicht zulässt, sondern im Kontext der Situation und des bisherigen Bearbeitungsverlaufs verstanden werden muss.

Inhaltlich zeigt sich, dass Lehrpersonen Feedback nicht allein rückblickend im Sinne von *feed back* formulieren, sondern häufig zukunftsgerichtet als *feed forward* gestalten (Hattie und Timperley 2007). Anstatt lediglich auf bestehende Fehler hinzuweisen, geben sie Hinweise darauf, wie Lernende ihr weiteres Vorgehen planen oder verbessern können. Dieses handlungsorientierte Feedback unterscheidet sich deutlich von dem meist reaktiven Charakter automatisierter Systeme, die primär auf Fehlermeldungen, Testfälle oder statische Codeanalysen reagieren. Während solche Systeme vor allem syntaktische und technische Aspekte adressieren, zielen Lehrpersonen auf kognitive, strategische und motivationale Lernprozesse ab. Damit spiegelt das HOW-Material eine pädagogische Zielsetzung wider, die über die reine Fehlerbehebung hinausgeht und die Entwicklung fachlicher und metakognitiver Kompetenzen in den Blick nimmt.

Besonders hervorzuheben ist, dass motivationale Elemente zwar nicht als eigenständiger Auslöser (`WHY`) identifiziert wurden, jedoch als charakteristisches Merkmal in der sprachlichen Gestaltung des Feedbacks auftreten. Bestätigende, ermutigende oder wertschätzende Formulierungen dienen dabei als didaktische Rahmung, die das Vertrauen der Lernenden stärkt und ihre Bereitschaft zur Weiterarbeit unterstützt. Diese Beobachtung verweist auf eine klare funktionale Trennung zwischen dem Grund für ein Eingreifen (`WHY`) und der Art der Ausgestaltung (`HOW`) einer Rückmeldung.

Insgesamt unterstreichen die Ergebnisse, dass sich das Feedback erfahrener Lehrpersonen durch eine hohe Kontextsensitivität, kommunikative Vielfalt und strategische Zielorientierung auszeichnet. Im Gegensatz zu automatisierten Systemen, die auf festgelegte Auslöser und formale Fehlerstrukturen reagieren, reflektiert menschliches Feedback eine didaktisch begründete Balance zwischen Unterstützung und Selbststeuerung. Für die Modellierung adaptiver Lernumgebungen bedeutet dies, dass Feedbackregeln nicht nur inhaltlich (z. B. Art des Fehlers), sondern auch kommunikativ (z. B. Form der Ansprache, Grad der Offenheit) differenziert beschrieben werden müssen. Damit liefert die HOW-Analyse einen zentralen Beitrag zur formalen Abbildung von Feedback-Strategien und bildet die Grundlage für kontextsensitive, lernförderliche Rückmeldesysteme.

4.6 Limitationen

Die folgenden Einschränkungen betreffen beide Analysen (`WHY` und `HOW`) und beziehen sich auf die zugrunde liegende Methodik, das Datenmaterial sowie die Ableitung und Anwendung der Kategoriensysteme. Sie sollen die Reichweite der Befunde einordnen und Ansatzpunkte für weiterführende Untersuchungen aufzeigen.

Datenbasis und Aufgabenraum. Die Untersuchung basiert auf einer begrenzten Zahl von Bearbeitungssequenzen, die vorwiegend typische Anfängeraufgaben mit funktionalem bzw. mathematischem Zuschnitt umfassen. Dadurch wird der Aufgabenraum eingegrenzt, und es bleibt offen, inwieweit die abgeleiteten Entscheidungs- und Realisierungslogiken auch für andere Programmiersprachen, Aufgabentypen oder Lernendengruppen gelten. Zudem war die vollständige Rekonstruktion der situationsspezifischen Kontexte nicht in allen Fällen möglich: Die Vorverarbeitung der Sequenzen wurde durch unterschiedliche Systemversionen der eingesetzten Lernumgebung erschwert, sodass nicht abschließend sichergestellt werden konnte, dass die auf Screenshots sichtbaren Systemrückmeldungen exakt denjenigen entsprachen, die die Lernenden im Bearbeitungszeitpunkt erhielten.

Reaktivität und Selbstauskunft. Das Studiendesign beruht auf schriftlichen Einschätzungen und Rückmeldungen erfahrener Lehrpersonen zu vorgegebenen Situationen. Damit sind typische Einschränkungen der Selbstauskunft verbunden, etwa Einflüsse sozialer Erwünschtheit, selektive Wahrnehmung oder Auslassungen. Darüber hinaus kann nicht ausgeschlossen werden, dass das Verhalten der Lehrpersonen durch das Bewusstsein der Beobachtung beeinflusst wurde – ein Effekt, der als *Observer's Paradox* beschrieben wird (Roethlisberger et al. 2003). Dieser besagt, dass Teilnehmende ihr Verhalten oder ihre Äußerungen verändern, sobald sie wissen, dass sie Gegenstand einer Untersuchung sind. Entsprechend ist denkbar, dass einige Rückmeldungen stärker normativ oder idealtypisch formuliert wurden, als dies in realen Unterrichtssituationen der Fall wäre. Hinzu kommt, dass keine multimodalen Lernendendaten (z. B. Verbalisierungen, Gestik, Mimik) vorlagen, die in authentischen Lernsituationen häufig zusätzliche Signale für Feedbackentscheidungen darstellen. Einzelne Lehrpersonen richteten ihre Antworten zudem an die Forschenden statt an die Lernenden, wodurch unklar blieb, ob es sich um hypothetische Überlegungen oder tatsächlich intendierte Rückmeldungen handelte.

Kodierentscheidungen und Kategorienabgrenzung. Die Entwicklung und Anwendung der Kategoriensysteme folgte klaren Kodierregeln, erforderte jedoch interpretative Entscheidungen, insbesondere in Fällen, in denen mehrere Kategorien gleichzeitig zutrafen oder konzeptuell überlappten. So war etwa in der `WHY`-Analyse die Abgrenzung zwischen `PROGR` und `TRUST` nicht immer eindeutig: Während `PROGR` typischerweise Situationen beschreibt, in denen ein laufender Bearbeitungsprozess beobachtet und bewusst nicht unterbrochen wird ("Lassen Sie uns die Entwicklung der Situation beobachten"), steht `TRUST` für eine antizipierende Einschätzung der Lernenden, etwa im Sinne eines zugeschriebenen Potenzials zur selbstständigen Fehlerbehebung ("Ich bin zuversichtlich, dass sie diesen Fehler von selbst erkennen wird"). In manchen Fällen waren beide Lesarten gleichzeitig plausibel, wenn Fortschritt und zugeschriebene Kompetenz zusammentrafen. Auch in der HOW-Analyse traten ähnliche Unschärfen auf, vor allem bei sehr knapp formulierten Rückmeldungen. So war nicht immer erkennbar, ob sich eine Aussage auf die konkrete Repräsentation eines Fehlers im Code oder auf die systemseitige Fehlermeldung der Lernumgebung bezog (`EXFEED`). Ein Beispiel ist das Feedback "Start with the error message", das je nach Kontext als `KM-CE-`Ⓘ (Information zum Fehler), als `KM-CE-`Ⓡ (Anstoß zur Reflexion) oder als `KTC-TPR-`Ⓘ (strategischer Handlungsimpuls) interpretiert werden kann. Solche Mehrdeutigkeiten verdeutlichen die Komplexität realer Feedbackhandlungen, erschweren jedoch die Festlegung einer primären Kategorie. Auch multifunktionale Rückmeldungen – etwa die Kombination aus motivierenden Elementen (`MOT`) und prozeduralen Hinweisen (`KH`) – zeigen die enge Verflechtung verschiedener Feedbackfunktionen, was die Vergleichbarkeit über Fälle hinweg einschränkt.

System- und Kontextinformationen. Die Interpretation der Feedbacksituationen setzt voraus, dass die zugrunde gelegten Aufgabenkontexte und Systemmeldungen authentisch wiedergegeben sind. Versionswechsel oder geringfügige Unterschiede in der Lernumgebung (z. B. IDE oder LMS) könnten jedoch dazu geführt haben, dass Lehrpersonen auf leicht veränderte Auslöser reagierten. Ohne vollständige Logdaten oder Zugriff auf die tatsächlichen Interaktionsverläufe bleibt die Kontextrekonstruktion daher in Teilen auf Annahmen und Beobachtungen gestützt.

Adressierung und Intention. Für die HOW-Analyse war entscheidend, die Intention einer Äußerung – etwa als Leitfrage, Hinweis oder Information – präzise zu erfassen. Einige Lehrpersonen formulierten ihr Feedback jedoch metasprachlich oder ergänzten hypothetische Bedingungen („Wenn der/die Studierende X täte, dann ..."). In solchen Fällen ließ sich nicht immer klar bestimmen, ob die Äußerung als tatsächliches Feedback an Lernende oder als theoretische Überlegung zu verstehen war. Besonders bei kurzen, kontextarmen Aussagen hing die Interpretation stark vom situativen Umfeld ab, wodurch die eindeutige Zuordnung erschwert wurde.

Externe Validität. Die Befunde dieser Studie geben Einblick in professionelle Feedbackpraktiken unter kontrollierten Bedingungen. Sie zeigen, *was* erfahrene Lehrpersonen in den analysierten Situationen berichten und formulieren, erlauben jedoch keine direkten Rückschlüsse darauf, *wie* sie in realen Unterrichtssituationen handeln oder *welche* Strategien nachweislich lernwirksam sind. Eine Validierung der Befunde unter authentischen Bedingungen – etwa in Form von Feldstudien oder Prozessanalysen mit lautem Denken – wäre daher ein wichtiger nächster Schritt, um die Wirksamkeit und Generalisierbarkeit der identifizierten Muster zu prüfen.

Kapitel 5

Formalisierung von Aufgaben und Antworten

Die bisherigen Analysen haben verdeutlicht, dass Feedbackentscheidungen von Lehrpersonen nicht ausschließlich auf der inhaltlichen Korrektheit von Antworten beruhen, sondern in hohem Maße auch vom Aufgabenkontext, dem Bearbeitungsverlauf und weiteren situativen Bedingungen geprägt sind (vgl. Kapitel 4). Lehrpersonen berücksichtigen dabei sowohl produktbezogene Merkmale wie Korrektheit, Codequalität und Effizienz als auch prozessbezogene Indikatoren wie Fortschritt, Pausen oder Strategiewechsel. Hinzu kommen interpretative Einschätzungen zur Selbstständigkeit, Motivation oder Unsicherheit der Lernenden, die das Eingreifen zusätzlich steuern.

Diese Vielschichtigkeit der Entscheidungsgrundlagen macht deutlich, dass Feedbackprozesse auf einer komplexen, kontextsensitiven Informationsbasis beruhen. Soll ein vergleichbarer Grad an Kontextsensitivität auch in automatisierten Lernumgebungen realisiert werden, erfordert dies eine präzise, maschinenlesbare Modellierung sowohl der Aufgabenstrukturen als auch der typischen Antwortformen. So lassen sich die Voraussetzungen schaffen, unter denen Feedback nicht nur korrekt, sondern auch lernförderlich und individualisiert bereitgestellt werden kann.

Vor diesem Hintergrund widmet sich das folgende Kapitel der Frage, wie Aufgaben und Antworten so formalisiert werden können, dass sie als Grundlage für kontextsensitive Feedback-Strategien dienen. Dabei werden theoretische und empirische Erkenntnisse zusammengeführt, um zentrale Komponenten der Aufgabenmodellierung zu bestimmen und den Übergang von der didaktischen Analyse zur formalen Beschreibung vorzubereiten.

Zur Veranschaulichung der im Folgenden entwickelten Konzepte wird ein durchgängiges Beispiel herangezogen. Das in Abbildung 5.1 dargestellte Aufgabenbeispiel dient im weiteren Verlauf dieses Kapitels als *Running Example*.

Anhand dieser exemplarischen Programmieraufgabe wird schrittweise gezeigt, wie sich zentrale Aufgabenmerkmale mithilfe des Y-Modells erfassen und strukturieren lassen – etwa Bearbeitungsanforderungen, intendierte Lernziele, typische Fehlermuster oder charakteristische Antwortmuster – und wie diese Modellierung zur Generierung adaptiven Feedbacks beitragen kann. Das Beispiel fungiert somit als didaktischer Referenzpunkt, um die theoretische Argumentation mit einem realistischen Aufgabenformat zu verbinden und den Prozess der Formalisierung nachvollziehbar zu illustrieren.

Teile dieses Kapitels wurden bereits vorab veröffentlicht in Lohr et al. (2025c), Lohr et al. (2023) sowie Lohr und Berges (2025). Für Details siehe Übersicht zu Vorabveröffentlichungen auf Seite VII.

minSearch

Gegeben sei der folgende (leere) Klassenrumpf `Homework3` in Java:

```
public class Homework3 {
    // Code here
}
```

Implementieren Sie eine Methode `minSearch`, die ein Feld (`int[]`) von Ganzzahlen als Eingabe erhält und das kleineste darin enthaltene Element zurückgibt.

Abbildung 5.1: Running Example: exemplarische Programmieraufgabe zur Illustration der Aufgaben- und Antwortdimension im Y-Modell

5.1 Komponenten von Programmieraufgaben und Modellierungsansätze

Ausgehend von den in Abschnitt 2.2 dargestellten Grundlagen, insbesondere der Definition von Ruf, Berges und Hubwieser (2015), sowie den empirischen Befunden zur Feedbackpraxis (Kapitel 4), lassen sich vier zentrale Komponenten identifizieren, die für die strukturierte Modellierung von Aufgaben in adaptiven Programmierlernsystemen besonders relevant erscheinen:

1. **Wissen**: Aufgaben sind immer mit bestimmten Wissensvoraussetzungen verbunden. Diese betreffen sowohl fachliches deklaratives Wissen (z. B. Syntaxregeln, Methodenaufrufe) als auch prozedurales Wissen über Vorgehensweisen oder Lösungsstrategien. In der Analyse der Feedbackpraxis zeigte sich, dass Lehrpersonen beim Geben von Rückmeldungen häufig implizit prüfen, ob relevantes Vorwissen aktiviert oder vorhanden ist, bevor sie bestimmte Hinweise geben (Kapitel 4). Außerdem bezogen sich einige Rückmeldungen konkret auf die Erklärung von Konzepten (`KC`-Feedback). Die Modellierung von (Fach-)Wissen stellt somit eine grundlegende Komponente für die Generierung kontextsensitiver Rückmeldungen dar.

2. **Kognitive Prozesse**: Zur Bearbeitung einer Aufgabe müssen Lernende spezifische Denkprozesse durchlaufen – von der Reproduktion über das Anwenden bis hin zum Problemlösen oder Reflektieren. Diese Prozesse bestimmen maßgeblich den Schwierigkeitsgrad und die Art der geforderten kognitiven Leistung. In den Feedbackentscheidungen der Lehrpersonen wurde deutlich, dass Rückmeldungen häufig auf das beobachtbare Bearbeitungsverhalten in unterschiedlichen Phasen Bezug nahmen (z. B. Pausen, Strategiewechsel, fehlerhafte Annahmen). Dies verweist auf eine implizite Einschätzung kognitiver Anforderungen durch die Lehrpersonen.

3. **Aufgabenstellung und Kontext**: Die Art und Weise, wie eine Aufgabe formuliert und gerahmt ist, beeinflusst das Bearbeitungsverhalten erheblich. Vorgaben wie die verpflichtende Nutzung bestimmter Konzepte – etwa Rekursion bei der Fibonacci-Aufgabe (vgl. Tabelle 4.2) – begrenzen den Lösungsraum und lenken die Wahl der Bearbeitungsansätze. Strukturelle Merkmale wie Beispiele, formale Einschränkungen oder sprachliche Hinweise wirken ebenfalls steuernd. In den Analysen zeigte sich, dass erfahrene Lehrpersonen häufig Rückmeldungen zu solchen Merkmalen gaben (`KTC`-Feedback) – vermutlich um den Bearbeitungsprozess gezielt zu steuern. Die Modellierung dieser kontextuellen Faktoren ist daher zentral, um automatisiertes Feedback aufgabenangemessen und situativ fundiert gestalten zu können.

4. **Antworten**: Der Zielzustand einer Aufgabe manifestiert sich in der von den Lernenden gegebenen Antwort. Diese kann korrekt, unvollständig oder fehlerhaft sein, liefert aber stets Hinweise auf Denkwege, Missverständnisse oder alternative Lösungsstrategien. Die in

Kapitel 4 dargestellten Ergebnisse zeigen, dass nicht nur die finale Korrektheit einer Lösung, sondern auch deren Struktur und Qualität (z. B. Stil, Vorgehensweise, Klarheit) für Feedbackentscheidungen relevant sind. Eine explizite Modellierung typischer Antwortformen eröffnet daher differenzierte diagnostische Zugänge.

Die Analyse dieser vier Komponenten bildet die Grundlage für eine systematische Bestimmung der relevanten Aufgabenelemente und deren spätere Formalisierung (vgl. Abschnitt 3.2, **FF2.1**). In den folgenden Abschnitten werden sie jeweils einzeln betrachtet, theoretisch fundiert und mit Bezug auf einschlägige Modelle aus der Literatur konkretisiert. Dabei steht im Vordergrund, wie sich diese Dimensionen so beschreiben lassen, dass eine algorithmische Interpretation von Aufgaben und Bearbeitungsverläufen möglich wird. Die hier entwickelten Einsichten sind jedoch noch nicht als endgültige Formalisierung zu verstehen, sondern liefern die *Implikationen für die Aufgabenmodellierung*, die im Rahmen von Abschnitt 5.3 in einem integrativen Rahmen explizit formalisiert und zusammengeführt werden.

5.1.1 Wissenskomponente: Repräsentation von Domänenwissen

In der Definition von Ruf, Berges und Hubwieser (2015) wird eine Aufgabe als Transformation eines gegebenen Ausgangszustands in einen angestrebten Zielzustand beschrieben. Diese Transformation erfordert Wissen zur Durchführung der Handlung und ist zugleich mit bestimmten Wissensbeständen im Ausgangs- und Zielzustand verknüpft. Damit wird deutlich, dass Aufgaben immer explizit oder implizit auf (fachliche) Wissensinhalte und kognitive Strukturen Bezug nehmen.

Auch die in Kapitel 4 beschriebenen Ergebnisse zeigen, dass Feedbackentscheidungen von Lehrpersonen maßgeblich davon abhängen, welches Wissen Lernende bereits aufgebaut haben und welche Konzepte sie in einer Aufgabe anwenden sollen. Um diese kontextabhängigen Entscheidungsprozesse in formale Strukturen überführen zu können, erfordert die Modellierung von Aufgaben und Antworten eine explizite Beschreibung der zugrunde liegenden Wissenselemente. Nur wenn eindeutig erfasst ist, welches Wissen für die Bearbeitung erforderlich ist und welches im Lernprozess aufgebaut werden soll, können Bearbeitungsverläufe zuverlässig interpretiert und darauf aufbauend gezielte Rückmeldungen generiert werden. Dies gilt insbesondere für Feedbackformen wie *Knowledge about Concepts (KC)*, die auf einzelne fachliche Konzepte Bezug nehmen (siehe Kapitel 4). Ohne eine präzise Modellierung der Konzepte und ihrer Abhängigkeiten bestünde die Gefahr, dass Lernende Rückmeldungen zu Inhalten erhalten, die (noch) nicht zugänglich oder nicht relevant sind.

5.1.1.1 Wissensarten und ihre Rolle im Lernprozess

Der Begriff *Wissen* wird in verschiedenen Disziplinen unterschiedlich gefasst. Während erkenntnistheoretische Perspektiven meist die Wahrheit und Begründung von Aussagen voraussetzen (Gettier 1963), steht in der Bildungsforschung meist die Unterscheidung zwischen *Kompetenz* und *Wissen* im Vordergrund: *Wissen* beschreibt hier die Repräsentation von Informationen im Gedächtnis, während Kompetenzen deren situationsgerechte Anwendung umfassen (Weinert 2001; Anderson 2005). Für die Modellierung von Wissen im Rahmen dieser Arbeit wird ein pragmatischer Wissensbegriff zugrunde gelegt: Als *Wissen* gelten alle fachlich relevanten Inhalte, Konzepte, Regeln oder Strategien, die zur Bearbeitung einer Aufgabe erforderlich sind oder durch sie vermittelt werden sollen. Dabei ist zunächst unerheblich, ob dieses Wissen formal korrekt, vollständig oder subjektiv internalisiert ist (Kerres und de Witt 2004).

Die Bildungspsychologie unterscheidet verschiedene Arten von Wissen, die für Lernprozesse unterschiedlich relevant sind. Besonders verbreitet ist die Taxonomie von Anderson und Krathwohl (2001), die vier zentrale Wissensarten benennt (Anderson und Krathwohl 2001):

- **Deklaratives Wissen**: Fakten, Begriffe, syntaktische Regeln (z. B. „Wie wird eine Methode in Java definiert?“)
- **Konzeptuelles Wissen**: Zusammenhänge, Modelle, Prinzipien (z. B. „Was ist der Unterschied zwischen Referenz und Wert?“)
- **Prozedurales Wissen**: Strategien, Abläufe, Problemlösungsmethoden (z. B. „Wie iteriere ich über ein Feld und finde das Minimum?“)
- **Metakognitives Wissen**: Wissen über das eigene Denken und Lernen, etwa zur Planung, Überwachung und Regulation kognitiver Prozesse.

Für die hier verfolgte Formalisierung sind vor allem die ersten drei Wissensarten relevant, da sie sich direkt mit fachlichen Inhalten und Bearbeitungsschritten verknüpfen lassen. Metakognitives Wissen ist demgegenüber meist nur indirekt, etwa über Verhaltensdaten, erschließbar (Nkambou, Bourdeau und Mizoguchi 2010).

Die Programmieraufgabe in Abbildung 5.1 verdeutlicht dies exemplarisch: Sie erfordert deklaratives Wissen über Methodensignaturen in `Java`, konzeptuelles Wissen über die Struktur und Indizierung von Feldern sowie prozedurales Wissen zur Implementierung einer iterativen Suche. Diese Verbindung unterschiedlicher Wissensarten ist dabei nicht auf das dargestellte Beispiel beschränkt, sondern charakteristisch für Programmieraufgaben im Allgemeinen, da das Programmieren selbst auf der Integration verschiedener Wissensbestände beruht (Kiesler 2024; McGill und Volet 1997). Lernende müssen nicht nur fachliche Konzepte verstehen, sondern diese zugleich in Handlungssequenzen überführen und in einer formalen Sprache korrekt ausdrücken. Selbst einfache Aufgaben – etwa das Umsetzen einer bedingten Wiederholung oder die Implementierung einer mathematischen Berechnung – aktivieren typischerweise mehrere Wissensarten gleichzeitig: Sie setzen deklaratives Wissen über Syntax, konzeptuelles Wissen über Datentypen und Operatoren sowie prozedurales Wissen über Kontrolllogik voraus. Programmieraufgaben sind somit grundsätzlich als *mehrdimensionale Wissensanforderungen* zu verstehen, bei denen deklarative, konzeptuelle und prozedurale Elemente in wechselseitiger Abhängigkeit stehen.

5.1.1.2 Wissensrepräsentation in digitalen Lernsystemen

Um Wissen so zu repräsentieren, dass es maschinenlesbar verfügbar ist und für Diagnose- und Feedbackprozesse genutzt werden kann, wird in digitalen Lernsystemen häufig auf eine strukturierte Darstellung wie *Wissensgraphen* zurückgegriffen. In solchen Graphen werden Wissenselemente als Knoten und deren Beziehungen als Kanten dargestellt, wodurch sich Abhängigkeiten, Hierarchien oder semantische Verknüpfungen explizit modellieren lassen. Die genaue Ausgestaltung der Wissenselemente und Relationen kann dabei variieren (für einen aktuellen Überblick siehe (Ji et al. 2022)).

Wie in Abschnitt 2.3.3 bereits erläutert, greifen Intelligente Tutoring-Systeme auf unterschiedliche Paradigmen zurück, um solches Wissen systematisch nutzbar zu machen. Für Programmieraufgaben erscheinen insbesondere *Cognitive Models* relevant, da sie Lernprozesse auf der Ebene einzelner Wissenseinheiten erfassen und auf der Annahme beruhen, dass Lernen als sukzessiver Erwerb kleiner, strukturierter Wissenseinheiten erfolgt. Diese werden als *Knowledge Components* (KnoCs)[1] bezeichnet. Vanlehn (2006) definiert *KnoCs* als kleinste inhaltliche Einheiten – etwa Begriffe, Regeln oder Strategien –, die zur erfolgreichen Lösung einer Aufgabe erforderlich sind. Die aus Diagnose, Feedback und Lernfortschrittsverfolgung gewonnenen Informationen zu diesen Einheiten können im sog. *Lernenden-Modell* genutzt werden, um den Wissensstand einer lernen-

[1] Die in der Literatur übliche Abkürzung *KCs* wird hier zu *KnoCs* geändert, um eine Verwechslung mit der Feedbackkategorie *Knowledge of Concepts (KC)* zu vermeiden.

den Person abzuschätzen und daraus adaptive Entscheidungen abzuleiten (Corbett und Anderson 1995). Im Lernenden-Modell kann dann z. B. für jede *KnoC* ein individueller „Lernstand" geschätzt werden, der anzeigt, in welchem Maße die entsprechende Wissenseinheit bereits beherrscht wird. Auf dieser Grundlage lassen sich verschiedene adaptive Mechanismen realisieren, die das Lernsystem gezielt an den aktuellen Lernstand anpassen:

- **Adaptive Aufgabenauswahl:** Aufgaben können so gewählt oder generiert werden, dass sie gezielt jene *KnoCs* adressieren, die noch nicht hinreichend gefestigt sind.
- **Personalisierte Lernpfade:** Sequenzen und Schwierigkeitsgrade lassen sich dynamisch anpassen, um individuelle Lernverläufe optimal zu unterstützen.
- **Kontextsensitive Feedbackgenerierung:** Rückmeldungen können auf spezifische *KnoCs* verweisen, deren Beherrschung (noch) unsicher ist, und so gezielte Hilfen oder erklärende Hinweise bereitstellen.

Damit bilden *KnoCs* eine zentrale Schnittstelle zwischen Diagnostik, Aufgabensteuerung und Feedbackprozess. Sie ermöglichen es adaptiven Lernsystemen, Lernfortschritt nicht nur zu messen, sondern daraus unmittelbar individualisierte und kontextangepasste Unterstützungsmaßnahmen abzuleiten.

Forschungsarbeiten zeigen, dass sich insbesondere Programmierdomänen erfolgreich zur Modellierung mit *KnoCs* eignen, da sie durch eine hohe strukturelle Formalität und klare Regelhaftigkeit gekennzeichnet sind (Sampaio De Alencar et al. 2025). Die zugrunde liegende Syntax und Semantik der Programmiersprachen definiert präzise, welche Operationen, Datentypen und Kontrollstrukturen zulässig sind und wie diese kombiniert werden dürfen. Dadurch lassen sich Aufgaben und Lösungen in wohldefinierte, voneinander abgrenzbare Wissenselemente zerlegen, die spezifische kognitive oder fachliche Anforderungen repräsentieren. Zudem liegt in der Programmierung eine enge Kopplung zwischen konzeptuellem Wissen (z. B. Wiederholungen, Bedingungen, Methodenaufrufe) und beobachtbarem Verhalten (z. B. Programmoutput oder Fehlerzustände) vor. Diese Verbindung erleichtert die empirische Erfassung, Zuordnung und Überprüfung einzelner *KnoCs*, da Lernergebnisse anhand der Programmstruktur und -ausführung nachvollzogen werden können.

Dass Programmieraufgaben meist aus wiederkehrenden, modularen Bausteinen bestehen – etwa Kontrollstrukturen, Datenoperationen oder Funktionsaufrufen – begünstigt die Nutzung von *KnoCs* zusätzlich: die Modularität erlaubt es, *KnoCs* domänenspezifisch zu definieren und dennoch über Aufgaben hinweg vergleichbar zu halten. Beispiele für typische *KnoCs* in diesem Kontext sind etwa die korrekte Implementierung einer Methodenstruktur in Java, der Zugriff auf Elemente eines Feldes, die Anwendung einer Vergleichslogik oder die Rückgabe eines Werts. Jede dieser Einheiten bildet ein klar identifizierbares Wissenselement, das sich unabhängig beobachten, diagnostizieren und gezielt durch Feedback adressieren lässt.

Insgesamt schafft die formale und zugleich hierarchisch strukturierte Natur der Programmierung somit ideale Bedingungen, um Lernprozesse auf Ebene einzelner *KnoCs* zu modellieren und adaptive Mechanismen darauf aufzubauen.

Implikationen für die Aufgabenmodellierung

Für die Gestaltung von Lernaufgaben bedeutet dies: Die mit der Aufgabe verbundenen Wissenselemente sollten möglichst präzise bestimmt werden – sowohl hinsichtlich der Inhalte, die für die Bearbeitung vorausgesetzt werden, als auch jener, die durch die Aufgabe aktiviert oder aufgebaut werden sollen. Für die formale Beschreibung von Aufgaben ergeben sich damit drei zentrale Fragen:

- **Welches Wissen wird für die Bearbeitung der Aufgabe vorausgesetzt?**
  Dies betrifft die fachlichen Vorbedingungen, ohne deren Beherrschung eine Lösung nicht möglich ist.
- **Welches Wissen ist in der Aufgabe selbst explizit oder implizit repräsentiert?**
  Die Wissensverankerung kann in Form von Regeltexten, Beispielcode, initialen Code-Gerüsten, grafischen Darstellungen oder anderen Repräsentationen erfolgen. Auch wenn Wissen nicht direkt genannt ist, kann es durch implizite Hinweise oder durch den Lösungsweg aktiviert werden.
- **Welches Wissen soll durch die Aufgabe aufgebaut, gefestigt oder differenziert werden?**
  Formulierte Lernziele definieren das Zielwissen, das mit der Aufgabenbearbeitung adressiert wird.

Die Art und Weise, wie Wissen in einer Aufgabe repräsentiert und verankert ist – etwa als expliziter Begriff, Beispielcode oder grafische Darstellung – bestimmt maßgeblich, wie Lernende auf dieses Wissen zugreifen und welche Strategien sie im Lösungsprozess anwenden. Unterschiedliche Darstellungsformen heben jeweils verschiedene Eigenschaften eines Konzepts hervor und lenken die Aufmerksamkeit der Lernenden auf unterschiedliche Aspekte des Lerngegenstands. Wie empirische Studien zeigen, beeinflusst diese Vielfalt an Repräsentationen nicht nur das Verständnis, sondern auch die Art und Qualität des Problemlöseverhaltens (Hahn und Klein 2025). Diese Zusammenhänge sind zentral für die Analyse von Lernendenantworten, da sich daraus ableiten lässt, in welchem Maße die gewählte Repräsentation einer Aufgabe die Interpretierbarkeit und Qualität der generierbaren Rückmeldungen bestimmt.

5.1.2 Kognitive Komponente

Um eine Aufgabe zu bearbeiten, müssen Lernende nicht nur über relevantes fachliches Wissen verfügen, sondern auch in der Lage sein, dieses in geeigneter Weise zu nutzen (Ruf, Berges und Hubwieser 2015). Kognitive Prozesse sind für Lehrpersonen zwar nicht direkt beobachtbar, doch lassen sich aus dem sichtbaren Bearbeitungsverhalten Lernender – etwa durch typische Fehler, gewählte Lösungswege oder die Art der Argumentation – Hinweise auf zugrunde liegende Denkprozesse ziehen. Lehrpersonen treffen ihre diagnostischen Urteile somit auf Basis beobachtbarer Indikatoren, aus denen sie den Grad des Verständnisses und die kognitive Tiefe der Bearbeitung inferieren (Shavelson und Stern 1981). Die Fähigkeit, solche Beobachtungen angemessen zu interpretieren und daraus fundierte Schlüsse über die Kompetenzen von Lernenden zu ziehen, gehört zum professionellen Handlungsrepertoire erfahrener Lehrpersonen. Sie bildet eine zentrale Voraussetzung für diagnostische Einschätzungen und die darauf aufbauende Gestaltung gezielter Rückmeldungen (Helmke 2022).

Damit Bearbeitungsverläufe in adaptiven Lernsystemen nicht nur hinsichtlich ihrer Korrektheit, sondern auch im Hinblick auf das zugrunde liegende Anspruchsniveau interpretiert werden können, ist eine explizite Modellierung der mit einer Aufgabe verbundenen kognitiven Anforderungen erforderlich. Auf dieser Grundlage lassen sich differenzierte Feedback-Strategien entwickeln, die den jeweiligen kognitiven Prozessen angemessen sind. So erfordern Aufgaben, die auf das bloße Wiedergeben von erlerntem Wissen abzielen, andere Formen der Rückmeldung als solche, die auf das Anwenden oder Reflektieren von Konzepten ausgerichtet sind (Shute 2008).

5.1.2.1 Begriffsklärung und theoretische Einbettung

In der Lernpsychologie wird Kognition häufig als Gegenbegriff zu behavioristischen Reiz-Reaktions-Modellen verstanden: Statt ausschließlich *äußeres* Verhalten zu betrachten, rücken kognitionspsychologische Ansätze die *internen* Verarbeitungsmechanismen in den Mittelpunkt der Analyse (Neisser 1967).

Der sogenannte *cognitive turn* im Bildungsbereich markierte einen Paradigmenwechsel, in dem Lernen nicht mehr primär als Verhaltensänderung, sondern als aktive, strukturierte Verarbeitung von Wissen verstanden wurde (Gardner 1987; Bruner 1996). Im Mittelpunkt stehen seither Prozesse wie Problemlösen, mentale Repräsentationen oder strategisches Denken, die das Verständnis und die Bearbeitung von Aufgaben maßgeblich beeinflussen können.

Auch wenn sich solche internen Prozesse nicht direkt beobachten lassen, eröffnen sich im Kontext adaptiver Lernsysteme dennoch Möglichkeiten, über das Bearbeitungsverhalten – etwa gewählte Lösungswege, typische Fehler oder Antwortmuster – Rückschlüsse auf zugrunde liegende kognitive Prozesse zu ziehen (Shavelson und Stern 1981; Südkamp und Praetorius 2017). Eine explizite Beschreibung kognitiver Anforderungen einer Aufgabe kann somit dazu beitragen, Lernverläufe differenzierter zu interpretieren und die Passung von Feedback zu verbessern (Anderson und Krathwohl 2001).

In Kombination mit der Wissenskomponente (vgl. Abschnitt 5.1.1.2) bietet die Modellierung kognitiver Prozesse damit eine zusätzliche Perspektive auf Lernprozesse: Während die Wissensdimension beschreibt, *welches Wissen* bei der Aufgabenbearbeitung adressiert wird, fokussiert die kognitive Dimension auf die Art und Weise, *wie* Lernende sich mit diesem Wissen auseinandersetzen. Diese Kombination eröffnet die Möglichkeit, Feedback nicht nur inhaltlich korrekt, sondern auch kognitiv angemessen zu gestalten.

5.1.2.2 Taxonomien als Instrument der Modellierung

Die systematische Modellierung kognitiver Prozesse ist ein zentrales Anliegen bildungspsychologischer Forschung seit den 1950er Jahren. Um Denkprozesse systematisch beschreiben und in Aufgabenmodellierungen berücksichtigen zu können, greift man häufig auf Klassifikationssysteme zurück: *Lerntaxonomien* ermöglichen es, kognitive Anforderungen von Lernaufgaben zu beschreiben, Lernziele zu definieren und Lernergebnisse vergleichbar zu machen (Fuller et al. 2007).

Für die Auswahl einer geeigneten Taxonomie zur Aufgabenmodellierung im Bereich der Programmierung sind insbesondere drei Kriterien bedeutsam. Sie ergeben sich aus dem grundlegenden Ziel adaptiver Lernsysteme, Lernprozesse individuell zu unterstützen und automatisch auf Lernvoraussetzungen zu reagieren:

- *Domänenspezifische Anschlussfähigkeit:* Sie sollte für Aufgaben im Bereich der Informatik bzw. Programmierung sinnvoll anwendbar sein.
- *Inhaltsunabhängigkeit:* Sie sollte kognitive Prozesse unabhängig vom konkreten fachlichen Wissen beschreiben können, um übertragbar zu bleiben.
- *Praktische Nutzbarkeit:* Sie sollte hinreichend verständlich und praktikabel sein, sodass sie von Lehrpersonen zur Annotation und Gestaltung von Aufgaben eingesetzt werden kann.

In der Hochschullehre haben sich insbesondere die Taxonomie von Bloom (1986) sowie ihre überarbeitete Version von Anderson und Krathwohl (Anderson und Krathwohl 2001) (AKT) etabliert. Beide Taxonomien bieten eine hierarchische Struktur kognitiver Anforderungen, die unabhängig vom konkreten Fachkontext angewendet werden kann. Die ursprüngliche Taxonomie nach Bloom unterscheidet sechs kognitive Stufen: (1) *Wissen*, (2) *Verständnis*, (3) *Anwendung*, (4) *Analyse*, (5) *Synthese* und (6) *Beurteilung*. Diese Stufen sind hierarchisch angeordnet, d. h. anspruchsvollere Denkprozesse bauen auf grundlegenderen auf. Bloom selbst weist jedoch darauf hin, dass diese Hierarchie nicht strikt zu verstehen ist (Anderson und Krathwohl 2001).

In der überarbeiteten Fassung durch Anderson und Krathwohl (2001) wurde die Dimension des kognitiven Prozesses neu strukturiert: die Reihenfolge der oberen Stufen wurde verändert und

die Begriffe sprachlich überarbeitet. Statt *Synthese* und *Beurteilung* finden sich nun die Begriffe *Erzeugen* (auch: *Erschaffen*) und *Evaluieren* (auch: *Beurteilen*), wobei *Erzeugen* als die anspruchsvollste Form kognitiver Leistung gilt. Zudem wurden sämtliche Prozesskategorien als aktive Verben statt als Substantive formuliert, um die Handlungsorientierung und die Beobachtbarkeit der Lernhandlungen stärker hervorzuheben. Diese Änderung hat nach Anderson und Krathwohl das Ziel, die Formulierung konkreter Lernziele sowie die didaktische Planung und Bewertung zu erleichtern (Anderson und Krathwohl 2001).

Auch die hierarchische Struktur wurde angepasst: Die bisherigen Stufen *Synthese* und *Evaluation* wurden durch die Begriffe *Erzeugen* und *Evaluieren* ersetzt und in ihrer Reihenfolge vertauscht. *Erzeugen* wird dabei als die anspruchsvollste Form kognitiver Leistung verstanden. Die sechs Dimensionen nach Anderson und Krathwohl (2001) lauten:

1. **Erinnern** – z. B. Wiedergeben von Fakten oder Definitionen
2. **Verstehen** – z. B. Interpretieren, Erklären, Umformulieren
3. **Anwenden** – z. B. Durchführen von Verfahren oder Routinen
4. **Analysieren** – z. B. Strukturieren, Differenzieren, Vergleichen
5. **Evaluieren/Beurteilen** – z. B. Begründen, Prüfen, Argumentieren
6. **Erzeugen/(Er-)schaffen** – z. B. Entwickeln neuer Lösungen oder Produkte

Eine Übersicht der ursprünglichen und der überarbeiteten Fassung, einschließlich der Verschiebung der oberen Stufen, zeigt Abbildung 5.2.

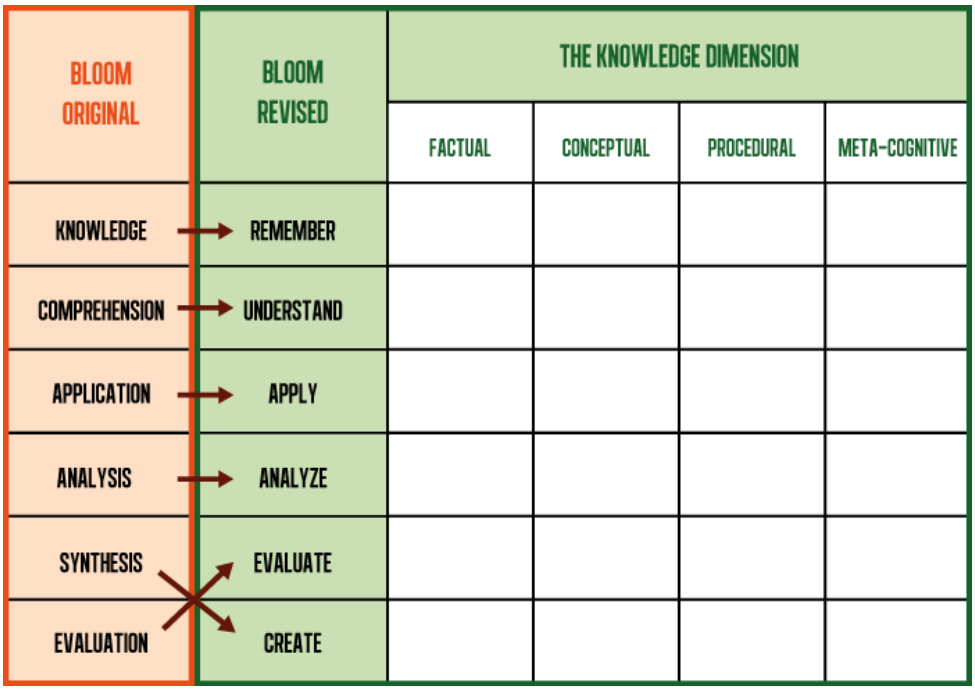


Abbildung 5.2: Vergleich der ursprünglichen Taxonomie von Bloom (1986) mit der überarbeiteten Taxonomie von Anderson et al. (1990)

Zudem ergänzen Anderson und Krathwohl (2001) die Taxonomie um eine zweite Dimension: die *Wissensdimension*. Sie unterscheidet deklaratives, konzeptuelles, prozedurales und metakognitives Wissen (vgl. Abschnitt 5.1.1.1). In Kombination mit der Prozessdimension entsteht so ein zweidimensionales Klassifikationsschema, das Lernziele und Aufgaben entlang zweier zentraler Fragen beschreibt: (1) *Welches Wissen* wird angesprochen? und (2) *Auf welchem kognitiven Niveau* soll es bearbeitet werden?

Für die Modellierung von Lernaufgaben in adaptiven Lernsystemen bietet die Taxonomie eine anschlussfähige Struktur: Sie ermöglicht es, kognitive Anforderungen unabhängig vom konkreten Fachinhalt zu beschreiben und mit spezifischen Wissenselementen zu verbinden. Durch ihre breite Akzeptanz in der Bildungsforschung und ihre Verankerung in der Ausbildung von Lehrkräften stellt sie zudem eine praktikable Grundlage dar – insbesondere deshalb, weil viele Lehrpersonen bereits mit dieser Taxonomie vertraut sind und sie aktiv in der Unterrichtsplanung oder Lernzielbestimmung einsetzen.

Während allgemeine Lerntaxonomien eine bewährte Grundlage zur Beschreibung kognitiver Anforderungen darstellen, stoßen sie im fachspezifischen Kontext der informatischen Bildung teilweise an ihre Grenzen. So untersuchten Johnson und Fuller (2006), inwieweit sich bestehende Lerntaxonomien für die Informatikausbildung eignen – etwa im Hinblick auf ihre Anwendung bei der Gestaltung von Kursen, Lehrmaterialien, Bewertungsprozessen oder beim Monitoring des Lernfortschritts.

Die Studie identifiziert zentrale Schwächen hierarchisch strukturierter Taxonomien wie derjenigen von Bloom oder der AKT – insbesondere in praxisnahen Disziplinen wie der Programmierung. Johnson und Fuller (2006) kommen zu dem Schluss, dass sich die einzelnen Stufen kognitiver Anforderungen im Programmierkontext häufig nicht klar voneinander abgrenzen lassen. So kann es beispielsweise leichter sein, Wissen zur Lösung einfacher Probleme anzuwenden, als dieses Wissen explizit zu beschreiben (Johnson und Fuller 2006; Berges, Mühling und Hubwieser 2012).

Vor diesem Hintergrund entwickelten Fuller et al. (2007) eine speziell auf den Kontext der Informatik zugeschnittene Taxonomie. Ihre Forschungsergebnisse legen nahe, dass sich kognitive Anforderungen beim Programmieren nicht immer entlang linear-hierarchischer Modelle beschreiben lassen. Insbesondere rezeptive (verstehende) und produktive (anwendende) Prozesse verlaufen in der Informatik häufig unabhängig voneinander und können nicht eindeutig in eine einheitliche Abfolge gebracht werden (Johnson und Fuller 2006; Berges, Mühling und Hubwieser 2012). Aufbauend auf der kognitiven Dimension der revidierten Bloom-Taxonomie nehmen sie daher eine grundlegende strukturelle Anpassung vor: Anstatt kognitive Anforderungen entlang einer einzigen hierarchischen Abfolge zu organisieren, werden diese in zwei orthogonalen, also voneinander unabhängigen, Dimensionen aufgeteilt (siehe Abbildung 5.3).

Die horizontale Achse beschreibt dabei rezeptive Fähigkeiten, also das Verstehen und Interpretieren bestehender Produkte. Die vertikale Achse erfasst produktive Prozesse, etwa das Entwerfen und Implementieren neuer Produkte. Die untersten Ebenen befinden sich in der linken unteren Ecke, und es versteht sich, dass die Studierenden sie von links nach rechts bzw. von unten nach oben durchlaufen. Laut Fuller et al. (2007) ist es beispielsweise nicht möglich, mit der Umgestaltung von Code zu beginnen, wenn noch nicht ein bestimmtes Kompetenzniveau im Sinne einer Anwendung von Programmcode erreicht wurde. Ein wesentlicher Vorteil dieser Matrix-Taxonomie besteht darin, dass sich verschiedene Lernpfade abbilden lassen. So kann eine Bewegung in horizontaler Richtung als vorwiegend theoretischer Kompetenzaufbau verstanden werden (Pfad 2 in Abbildung 5.3), während eine Bewegung in vertikaler Richtung den Erwerb praktischer Kompetenzen beschreibt (Pfad 1 in Abbildung 5.3). Diese Flexibilität macht die Taxonomie insbesondere für eine automatisierte Bereitstellung adaptiver Lernpfade in digitalen Lernsystemen interessant.

Implikationen für die Aufgabenmodellierung

Aus den vorangegangenen theoretischen und empirischen Überlegungen wird deutlich, dass für die automatisierte Bereitstellung kontextsensitiver Rückmeldungen nicht allein relevant ist, *welches Wissen* eine Lernaufgabe adressiert. Ebenso entscheidend ist, *welche kognitiven Prozesse* sie erfordert und *woran* sich diese im Bearbeitungsprodukt oder -verlauf erkennen lassen. Eine ex-

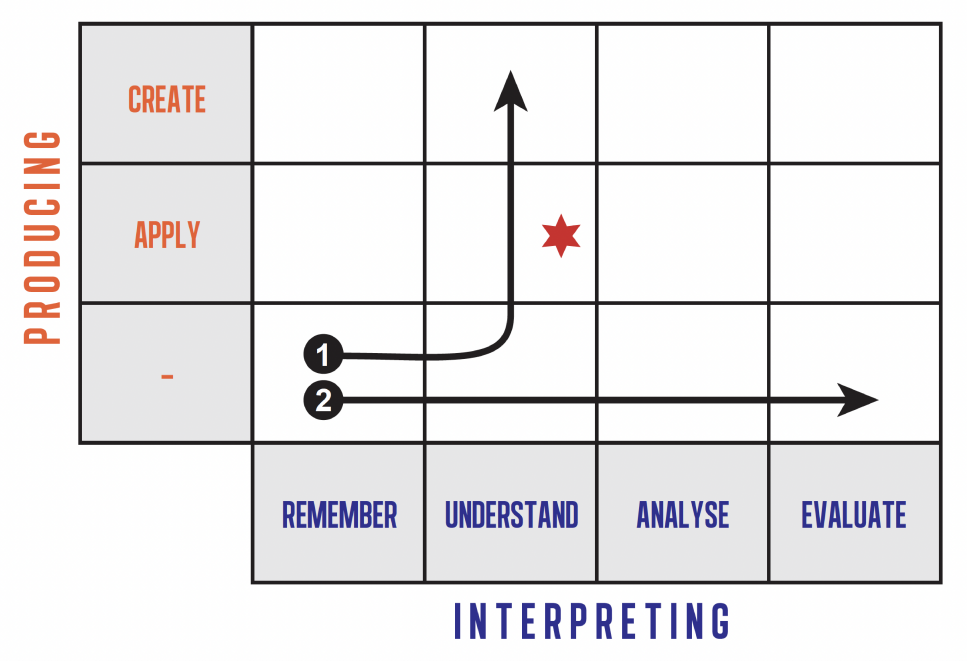


Abbildung 5.3: Zweidimensionale Matrix des kognitiven Modells nach Fuller et al. (2007)

plizite Modellierung der kognitiven Komponente schafft die Voraussetzungen dafür, beobachtete Antworten systematisch als Evidenz für Denkprozesse zu interpretieren und Feedback passgenau auszusteuern. Im Ergebnis muss die Aufgabenbeschreibung es erlauben, (i) kognitive Anforderungen einer Aufgabe zu bestimmen, (ii) aus Antworten valide Rückschlüsse auf das zugrunde liegende Anspruchsniveau zu ziehen und (iii) daraus regel- oder modellbasierte Feedbackentscheidungen abzuleiten.

Für die formale Beschreibung ergeben sich damit folgende zentrale Fragen:

- **Welche kognitiven Prozesse sind intendiert?**
  Zu spezifizieren ist das angestrebte Anspruchsniveau (z. B. Erinnern, Verstehen, Anwenden, Analysieren, Evaluieren, Erzeugen) sowie ggf. die Trennung von rezeptiven und produktiven Prozessen.
- **Welche *Evidenzen* machen diese Prozesse beobachtbar?**
  Festzuhalten sind prüfbare Indikatoren im Lösungsprodukt oder -prozess (z. B. Begründungen, Zwischenstände, Tests, Vergleiche, Refaktorierungsschritte), anhand derer eine Zuordnung zu kognitiven Prozessen entschieden werden kann.

Dabei ist zu beachten, dass sich das kognitive Anspruchsniveau nicht allein aus der Aufgabenstellung ableiten lässt. Welche Denkprozesse eine Aufgabe tatsächlich auslöst, hängt in hohem Maße von individuellen Faktoren wie Vorwissen, Erfahrungen und Bearbeitungsstrategien ab. Dieselbe Aufgabe kann für die eine Person eine reine Reproduktionsleistung darstellen, während sie für eine andere Transfer erfordert (Kleinknecht et al. 2013). Entsprechend zeigen Studien, dass selbst erfahrene Lehrpersonen Aufgaben ohne konkreten Bearbeitungskontext häufig unterschiedlich in kognitive Kategorien einordnen (Johnson und Fuller 2006).

Vor diesem Hintergrund wird in dieser Arbeit ein aufgabenfokussierter Zugang gewählt: Ausgangspunkt der Modellierung ist stets die intendierte kognitive Anforderung aus Sicht der Aufgabenstellung. Interindividuelle Unterschiede im Lernverhalten können ergänzend über Lernendenmodelle erfasst werden und bilden eine Perspektive für weiterführende Arbeiten.

Die vorgeschlagene Modellierung verknüpft benannte kognitive Anforderungen mit überprüfbaren Evidenzen im Produkt oder Prozess der Bearbeitung. Dadurch werden Antworten nicht nur hinsichtlich ihrer Korrektheit, sondern als Indikatoren zugrunde liegender Denkprozesse interpretierbar und Feedback kann *kognitiv angemessen* gesteuert werden. Die kognitive Komponente bildet damit die zweite Dimension neben der Wissenskomponente: Während diese beschreibt, *welches Wissen* adressiert wird, fokussiert die kognitive Komponente darauf, *wie* Lernende sich mit diesem Wissen auseinandersetzen.

5.1.3 Aufgabenstellung

Ausgehend von der Definition von Ruf, Berges und Hubwieser (2015) (vgl. Abschnitt 2.2.2), nach der eine Aufgabe als Transformation eines gegebenen Anfangszustands in einen angestrebten Zielzustand verstanden wird, stellt sich die Frage, wie diese Zustände – sowie die damit verbundenen kognitiven Prozesse – an die Lernenden kommuniziert werden. Genau hier setzt die *Aufgabenstellung* an: Sie fungiert als Medium, über das Anforderungen, Rahmenbedingungen und intendierte Bearbeitungspfade vermittelt werden (Merrill 2002).

Als erste Schnittstelle zwischen Lernenden und Aufgabe übernimmt die Aufgabenstellung eine zentrale didaktische Steuerungsfunktion. Sie transportiert wesentliche Informationen über die Zielsetzung der Aufgabe, die zu bearbeitenden Wissenselemente sowie das erwartete kognitive Anspruchsniveau. Damit beeinflusst sie maßgeblich, wie Lernende die Aufgabe verstehen, interpretieren und bearbeiten (Kleinknecht et al. 2013; Merrill 2002). Aus didaktischer Perspektive kann die Aufgabenstellung somit als Mechanismus verstanden werden, über den Anforderungen an Wissen, Kognition und Handlungsstrategien gezielt gestaltet und gesteuert werden.

Aufgabenstellungen definieren folglich nicht nur, *was* zu tun ist, sondern prägen in erheblichem Maße, *wie* Lernende an die Bearbeitung herangehen. Je nach Formulierung kann Vorwissen aktiviert, relevantes Wissen explizit oder implizit vorausgesetzt und die Ausführung bestimmter kognitiver Prozesse gezielt angeregt werden, die für die Lösung erforderlich sind. Dadurch beeinflusst die Aufgabenstellung sowohl die Auswahl von Bearbeitungsstrategien als auch die Struktur der resultierenden Lösungswege.

Für adaptive Lernumgebungen eröffnet eine präzise Modellierung der Aufgabenstellung weitreichende Potenziale: Sie ermöglicht es, Bearbeitungsprozesse differenzierter zu interpretieren, Lernschwierigkeiten frühzeitig zu erkennen und individualisiertes Feedback inhaltlich wie kognitiv passgenau bereitzustellen. Damit bildet die Aufgabenstellung ein zentrales Bindeglied zwischen Aufgabenstruktur, Lernendenverhalten und der Generierung kontextsensitiver Rückmeldungen.

5.1.3.1 Explizite und implizite Informationsträger

Nicht alle für die Bearbeitung relevanten Informationen sind in der Aufgabenstellung explizit genannt. In vielen Fällen enthalten Aufgabenstellungen sowohl explizite als auch implizite Hinweise – in Bezug auf fachliche Inhalte (z. B. zu verwendende Konzepte) ebenso wie auf kognitive Anforderungen (z. B. erwartete Denkprozesse oder Bearbeitungsstrategien).

Häufig werden bestimmte instruktionale oder situative Voraussetzungen stillschweigend vorausgesetzt – etwa, dass eine konkrete Programmiersprache genutzt wird, bestimmte API-Funktionen nicht erlaubt sind oder typische Sonderfälle berücksichtigt werden sollen. Solche impliziten Voraussetzungen ergeben sich z. B. aus dem Kurskontext, aus vorangegangenen Teil-(Aufgaben), begleitenden Instruktionen oder institutionellen Konventionen. Je nach Präzision und Ausgestaltung der Formulierung lassen sich diese Informationen mehr oder weniger eindeutig aus der Aufgabenstellung ableiten.

Zur weiteren Veranschaulichung wird im Folgenden das in Abbildung 5.1 eingeführte *Running Example* erneut aufgegriffen. Anhand zweier Varianten derselben Programmieraufgabe lässt sich zeigen, wie stark die Formulierung der Aufgabenstellung den Grad der expliziten Steuerung beeinflusst. Während die Aufgabenstellung der exemplarischen Programmieraufgabe in Abbildung 5.1 zentrale Wissenselemente und kognitive Anforderungen ausdrücklich benennt, zeigt Abbildung 5.4 hingegen eine Variante der Aufgabe mit stark verkürzter Aufgabenstellung, in der diese Elemente weitgehend implizit bleiben und erst durch den Kontext erschlossen werden müssen.

minSearch

Schreibe eine Methode, die das Minimum zurückgibt.

minSearch

Gegeben sei der folgende (leere) Klassenrumpf `Homework3` in Java:

```
public class Homework3 {
    // Code here
}
```

Implementieren Sie eine Methode `minSearch`, die ein Feld (`int[]`) von Ganzzahlen als Eingabe erhält und das kleinste darin enthaltene Element zurückgibt.

Abbildung 5.4: Knappe, implizite Aufgabenstellung im Vergleich zum Running Example

Der Vergleich verdeutlicht den Unterschied zwischen expliziten und impliziten Aufgabenformulierungen: Die in Abbildung 5.1 gezeigte *explizite Variante* benennt sowohl fachliche Wissenselemente (z. B. Felder, Kontrollstrukturen) als auch kognitive Anforderungen (z. B. `implementieren`, `beachten`) klar und lenkt damit den Lösungsraum stärker. Lernende erhalten eine eindeutige Vorstellung davon, welche Vorgehensweise erwartet wird. Die in Abbildung 5.4 dargestellte *implizite Variante* hingegen lässt zentrale Aspekte der Umsetzung offen oder setzt sie stillschweigend voraus – etwa das Einbinden in eine Java-Klasse, die Behandlung von Sonderfällen oder die zu verwendenden Sprachkonstrukte. Dadurch steigt die Bedeutung von Vorwissen und Kontextinformationen, da Lernende selbst erschließen müssen, welche Schritte erforderlich sind und welche Qualitätskriterien gelten.

Dieses Beispiel zeigt, dass Aufgabenstellungen nicht nur Informationen über das *Was*, sondern auch über das *Wie* der Bearbeitung transportieren. Für adaptive Lernsysteme stellt insbesondere die implizite Formulierung eine Herausforderung dar: Ohne ein zugrunde liegendes Kontextmodell sind solche stillschweigenden Anforderungen algorithmisch nur schwer zu erfassen. Eine formalisierte Aufgabenbeschreibung sollte daher – ergänzend zu den explizit genannten Angaben – auch jene impliziten Elemente modellieren, die für eine erfolgreiche Bearbeitung relevant sind, aber nicht ausdrücklich im Aufgabentext erscheinen. Dies betrifft sowohl fachliche Voraussetzungen und eingebettete Wissenselemente als auch intendierte kognitive Operationen, die für eine differenzierte Interpretation von Lernendenantworten entscheidend sind.

Die mit einer Aufgabe verbundenen Wissenselemente sind nicht immer explizit in der Aufgabenstellung benannt. Vielmehr können sie auf unterschiedliche Weise in die Aufgabenbeschreibung eingebettet sein: Sie erscheinen teils explizit als Fachbegriffe, teils implizit durch Verweise, Beispiele oder eingebettete Materialien wie Code-Snippets oder Diagramme. In manchen Fällen werden bestimmte Wissenselemente sogar erst durch die Aufgabenstellung selbst eingeführt – etwa durch erläuternde Kommentare oder begleitende Hinweise. Darüber hinaus gibt es auch Aufgaben, bei denen das relevante Wissen im Verlauf der Bearbeitung erst noch erschlossen oder erarbeitet werden muss. Diese Vielfalt an Darstellungsformen stellt eine Herausforderung für die

formalisierte Analyse dar, insbesondere wenn das Ziel darin besteht, Aufgaben systematisch mit den verbundenen Wissenselementen zu annotieren.

Auch die mit einer Aufgabe verbundenen kognitiven Anforderungen sind nicht immer explizit in der Aufgabenstellung formuliert. Sprachliche Operatoren wie *beschreiben*, *analysieren* oder *implementieren* können Hinweise darauf geben, welche kognitiven Prozesse durch die Aufgabe angeregt bzw. vorausgesetzt werden. Solche Operatoren lassen sich mit Stufen der kognitiven Dimension taxonomischer Modelle in Verbindung bringen (vgl. Abschnitt 5.1.2).

Zu unterscheiden ist dabei zwischen kognitiven Prozessen, die für die *Bearbeitung* erforderlich sind, und solchen, die durch die Aufgabe selbst *gefördert* werden. So kann eine Aufgabe mit dem Operator *implementieren* zugleich das *Analysieren* funktionaler Anforderungen oder das *Verstehen* zugrunde liegender Konzepte voraussetzen. Aufgaben-Operatoren können somit als *semantische Marker* für die Annotation kognitiver Anforderungen dienen und eine strukturierte Aufgabenmodellierung unterstützen.

Das Verständnis solcher Operatoren – also welche kognitiven Anforderungen mit welchem Begriff durch die Lehrperson adressiert werden – setzt auf Seiten der Lernenden prozedurales Wissen voraus. Es genügt nicht, die Begriffe auf deklarativer Ebene zu kennen – vielmehr ist Handlungswissen erforderlich, um bestimmte Denkoperationen auch praktisch umzusetzen zu können. Wer etwa eine Aufgabe mit dem Operator *analysieren* bearbeiten soll, muss wissen, wie man Inhalte systematisch zerlegt, Zusammenhänge erkennt und relevante Merkmale identifiziert. In der Taxonomie von Anderson und Krathwohl (2001) wird diese Unterscheidung explizit getroffen: während Aufgaben-Operatoren auf der kognitiven Prozessdimension verortet sind, gehört das Wissen über ihre angemessene Umsetzung zur prozeduralen Wissensdimension.

5.1.3.2 Einfluss der Formulierung auf Antwortverhalten

Die bisherigen Ausführungen machen deutlich: die sprachliche und strukturelle Gestaltung einer Aufgabenstellung hat unmittelbaren Einfluss auf das Antwortverhalten von Lernenden. Formulierungen, die etwa bestimmte Konzepte oder Lösungsstrategien explizit vorgeben – etwa die Nutzung von Rekursion oder die Einhaltung eines bestimmten Programmierstils – schränken den Lösungsraum gezielt ein. Ebenso können sprachliche Hinweise, Beispiele oder strukturelle Vorgaben (z. B. die Angabe von Methodennamen oder Parametertypen) das Vorgehen der Lernenden lenken. Die Art der Aufgabenstellung beeinflusst damit nicht nur die erwartete Lösung, sondern auch den Bearbeitungsverlauf und die zugrunde liegende kognitive Aktivität.

Diese Gestaltungsspielräume lassen sich gezielt didaktisch nutzen – etwa um bestimmte Denkprozesse anzuregen, typische Fehlvorstellungen aufzudecken oder schrittweise Problemlösestrategien zu fördern. Gleichzeitig birgt eine unpräzise oder missverständliche Formulierung das Risiko, dass Fehlinterpretationen entstehen, die sich in den Antworten niederschlagen – unabhängig vom eigentlichen fachlichen Verständnis. Für digitale Lernsysteme ist dies besonders bedeutsam, da Rückmeldungen in der Regel ausschließlich aus dem Antwortverhalten abgeleitet werden. Die Validität solcher diagnostischer Rückschlüsse hängt daher maßgeblich von der Eindeutigkeit und Transparenz der Aufgabenstellung ab.

Ein Ansatz zur Präzisierung besteht darin, Aufgaben systematisch als Übergänge zwischen einem definierten Ausgangszustand und einem angestrebten Zielzustand zu modellieren – wie es etwa in der Strukturierung nach Berges, Mühling und Hubwieser (2012) vorgeschlagen wird. Auch die ursprünglich sehr knappe Formulierung in Abbildung 5.4 lässt sich entsprechend erweitern: Tabelle 5.1 zeigt, wie der Aufgabeninhalt durch explizite Angabe von *gegebenem* und *gewünschtem* Zustand konkretisiert werden kann. Diese Klarheit ist nicht nur für die Lernenden hilfreich, sondern auch für die automatisierte Analyse und Interpretation von Antworten.

given state	Leerer Klassenrumpf der Klasse `Homework3`
desired state	Klasse `Homework3`, die eine Methode mit Namen `minSearch` enthält, die das Minimum eines übergebenen Feldes aus Ganzzahlen zurückgibt.

Tabelle 5.1: Aufgabenstellung mit expliziter Angabe von gegebenem und gewünschtem Zustand

Implikationen für die Aufgabenmodellierung Für die Modellierung von Aufgaben in adaptiven Lernsystemen ist es notwendig, die Aufgabenstellung als eigenständige, strukturierte Komponente zu erfassen. Sie bildet die Schnittstelle zwischen der intendierten Aufgabe und der Wahrnehmung durch die Lernenden und trägt entscheidend dazu bei, wie Wissen und kognitive Anforderungen im Bearbeitungsprozess wirksam werden.

Damit kontextsensitive Feedbackprozesse realisiert werden können, muss die Aufgabenstellung so modelliert sein, dass sowohl explizite als auch implizite Informationsträger identifizierbar und maschinenlesbar beschrieben werden können. Dazu zählen (i) fachliche Hinweise und Wissenselemente, die in der Formulierung oder in begleitenden Materialien enthalten sind, (ii) kognitive Anforderungen, die sich aus verwendeten Operatoren oder sprachlichen Strukturen ableiten lassen, sowie (iii) situative und instruktionale Kontexte, die das Aufgabenverständnis prägen.

Eine formalisierte Aufgabenbeschreibung sollte diese Aspekte in strukturierter Form erfassen – etwa durch semantische Annotationen, die erwartetes Vorwissen, intendierte Wissenselemente, kognitive Zielsetzungen oder relevante Kontexte ausweisen. Auf dieser Grundlage können Bearbeitungsverläufe algorithmisch interpretiert, typische Fehlermuster erkannt und Folgeaufgaben adaptiv ausgewählt werden.

Die Aufgabenstellung wird damit nicht nur als Träger didaktischer Intentionen verstanden, sondern als zentrale Steuergröße im adaptiven Lernprozess: Sie vermittelt zwischen der Wissens- und der kognitiven Dimension und schafft die strukturelle Grundlage dafür, Feedback anforderungsbezogen, lernförderlich und kontextsensitiv zu gestalten.

5.1.4 Antworten auf Aufgaben

Die Bearbeitung von Lernaufgaben liefert zentrale Informationen für die Regulation und Steuerung von Lernprozessen (Körndle, Narciss und Proske 2004). Die dabei entstehenden Antworten – als beobachtbare Ergebnisse des Bearbeitungsprozesses – bilden eine wesentliche Grundlage für Rückschlüsse auf zugrunde liegende, nicht direkt sichtbare kognitive Prozesse. Antworten zeigen, *wie* Lernende eine Aufgabe verstanden, interpretiert sowie bearbeitet haben, und erlauben damit Schlüsse auf Kompetenzniveaus, eingesetzte Strategien und mögliche Fehlkonzepte (Vanlehn 2006; Corbett und Anderson 1995). Auf dieser Basis sollte Feedback so gestaltet sein, dass eine nachvollziehbare Verbindung zwischen beobachtetem Verhalten und Rückmeldung entsteht und somit Reflexion über Denken und Handeln angeregt wird (Dainton 2018). Für die Entwicklung adaptiver Lernsysteme setzt dies voraus, dass Antworten automatisiert erfasst, interpretiert und mit den intendierten Zielzuständen der Aufgabe in Beziehung gesetzt werden können (Nkambou, Bourdeau und Mizoguchi 2010; Crow, Luxton-Reilly und Wuensche 2018).

5.1.4.1 Produktorientierung

Digitale Lernumgebungen und Intelligente Tutoring-Systeme zur Programmierausbildung fokussieren traditionell auf das *Endprodukt*, d. h. auf den eingereichten Programmcode. Im Zentrum stehen Korrektheit, Robustheit gegenüber Randfällen und – seltener – Aspekte der Codequalität

(Ihantola et al. 2010; Keuning, Jeuring und Heeren 2019). Diese *produktorientierte Sicht* greift in kompetenzorientierten Settings jedoch zu kurz, weil Kompetenzen nicht nur am Ergebnis, sondern insbesondere am *Weg dorthin* sichtbar werden (Weinert 2001). Das Problem verschärft sich im Kontext generativer KI, die auf Knopfdruck lauffähigen Code erzeugen kann (Finnie-Ansley et al. 2022): Allein am finalen Artefakt lässt sich immer schwerer zwischen Verständnis und bloßer Werkzeugnutzung unterscheiden. Folgerichtig ist eine Erweiterung der Analyse erforderlich, die auch Zwischenzustände und Bearbeitungsschritte berücksichtigt (Wiggins 2011). Im Folgenden werden zunächst *produktorientierte* Ansätze zur Modellierung von Antworten systematisch gebündelt. Ihre Stärken sowie Grenzen motivieren anschließend die prozessorientierte Ergänzung.

Produktorientierte Verfahren bewerten das Ergebnis einer Lernaktivität in Bezug auf den erwarteten Zielzustand (vgl. Abschnitt 5.1.3). Drei technische Linien sind in der Programmierausbildung besonders verbreitet (Ihantola et al. 2010; Keuning, Jeuring und Heeren 2019):

(1) Testbasierte Bewertung (dynamische Analyse). Eingereichter Code wird gegen definierte Testfälle ausgeführt – Korrektheit wird über bestandene Tests operationalisiert. Erweiterungen nutzen u. a. Grenz- und Negativtests zur Robustheitsprüfung sowie Abdeckungsmaße (Coverage) oder Mutationstests, um Testqualität einzuschätzen. Der Ansatz ist klar, objektivierbar und skalierbar. Grenzen zeigen sich bei *semantisch korrekten, aber didaktisch suboptimalen* Lösungen (z. B. ineffiziente Strategien) sowie bei Fällen, in denen Testfälle relevante Fehlkonzepte nicht sichtbar machen.

(2) Constraint-Based Modelling (CBM). CBM modelliert domänenspezifische Eigenschaften korrekter Lösungen als deklarative *Constraints* (Ohlsson 1994; Mitrovic 2012). Eine Lösung gilt als korrekt, sofern keine Constraint-Verletzung auftritt. Verletzungen markieren diagnostische Ankerpunkte und können mit gezielten Rückmeldungen verknüpft werden. CBM ist robust gegenüber Lösungsvielfalt, da nicht alle korrekten Wege explizit modelliert werden müssen, eignet sich damit auch für offene Aufgabenformate wie Programmieren und ist gut automatisierbar. Zugleich bleibt CBM auf Zustandsverletzungen beschränkt und erfasst weder Bearbeitungsstrategien noch die Genese der Lösung (Mitrovic 2010).

(3) Statische Code-Analyse. Bei der statischen Code-Analyse werden formale Eigenschaften des Quelltexts ohne Ausführung des Programms untersucht (z. B. über Abstract Syntax Trees). Typische Instrumente sind Linter und Stilprüfungen, Komplexitäts- und Strukturmetriken sowie Heuristiken zur Lesbarkeit oder Wartbarkeit. Diese Verfahren ergänzen test- oder CBM-basierte Bewertungen um Qualitätsaspekte des Endprodukts. Didaktisch relevante Unterschiede in der *Lösungsstrategie* (z. B. Wahl des Algorithmus) sind damit jedoch nur eingeschränkt erfassbar, sofern sie sich nicht in klaren statischen Merkmalen niederschlagen.

Diese drei Ansätze sind komplementär und bilden in der Praxis häufig kombinierte Pipelines: dynamische Tests zur funktionalen Korrektheit, CBM für deklarative Soll-Eigenschaften und statische Analyse für Qualitätsmerkmale (Ihantola et al. 2010; Keuning, Jeuring und Heeren 2019). Gemeinsamer Nenner ist die *Fokussierung auf das Endprodukt*. Dadurch bleiben wesentliche Aspekte kompetenzorientierter Diagnostik – etwa Strategiewahl, Irrwege, Strategiewechsel oder das Auftreten charakteristischer Zwischenzustände – weitgehend unsichtbar.

5.1.4.2 Prozessorientierung

Neben den produktorientierten Verfahren finden sich in der Literatur auch prozessorientierte Ansätze, die Bearbeitungsschritte und Zwischenzustände in den Fokus rücken. Verfahren wie

Model-Tracing oder Intention-Based Diagnosis beziehen Bearbeitungsverläufe explizit in die Analyse ein, indem sie Interaktionen wie Editieren, Kompilieren oder Testen protokollieren und mit idealisierten kognitiven Modellen vergleichen. Sie eröffnen tiefere Einblicke in Denkprozesse, Strategiewechsel und Fehlermuster. Gleichzeitig erfordern sie detaillierte Domänen- und Prozessmodelle, sind aufwendig zu kuratieren und reagieren empfindlich auf legitime, aber unerwartete Alternativwege; ihre Skalierbarkeit ist begrenzt (Crow, Luxton-Reilly und Wuensche 2018). In der Praxis dominieren daher weiterhin produktorientierte Assessments (Keuning, Jeuring und Heeren 2019) – mit den oben beschriebenen Grenzen, die sich durch den Einsatz generativer KI zusätzlich verschärfen (Finnie-Ansley et al. 2022).

Implikationen für die Aufgabenmodellierung

Für die Modellierung von Aufgaben bedeutet dies, dass auch mögliche Antwortformate und typische Antwortmuster explizit mitgedacht werden müssen. Die Aufgabenformalisierung umfasst daher nicht nur die Aufgabenstellung selbst, sondern auch ein strukturiertes Modell möglicher Reaktionen – inklusive der damit verbundenen Rückschlussmöglichkeiten. Damit rückt die Antwort nicht nur als Ergebnis, sondern als eigenständige modellierbare Komponente in den Fokus.

Eine zentrale Herausfordung bei der Analyse dieser Antworten besteht in der hohen Varianz: Insbesondere bei offenen Aufgabenformaten – wie etwa Programmieraufgaben – können sich Antworten selbst bei ähnlichem Kompetenzniveau der Lernenden stark unterscheiden. Diese Variabilität erschwert eine direkte Interpretation und erfordert intelligente Analyseverfahren.

Für die Modellierung von Programmieraufgaben folgt daraus:

1. **Antwortformate explizieren:** Neben der Aufgabenstellung sind die erwartbaren *Endprodukt-Merkmale* (funktionale Korrektheit, deklarative Soll-Eigenschaften, Qualitätskriterien) präzise zu beschreiben, um produktorientierte Analysen zuverlässig zu unterstützen.
2. **Diagnosepunkte festlegen:** Testfälle, Constraints und statische Metriken sollten so kuratiert werden, dass sie didaktisch relevante Fehl- und Teilzustände möglichst sichtbar machen (z. B. Randfälle, typische Misskonzepte).
3. **Erweiterung vorbereiten:** Da zentrale Kompetenzaspekte im Endprodukt nicht hinreichend erkennbar sind, ist eine prozessorientierte Ergänzung vorzusehen (vgl. Abschnitt 5.1.2). Sie ermöglicht, über das Endergebnis hinaus auch Zwischenzustände und Bearbeitungswege systematisch zu berücksichtigen.

5.2 Antwortklassen: ein didaktisch-diagnostisches Analysekonzept

Die zuvor beschriebenen Verfahren bieten wichtige methodische Zugänge, um die Vielfalt möglicher Lösungen zu strukturieren. Sie zeigen jedoch spezifische Einschränkungen hinsichtlich ihrer didaktischen Anschlussfähigkeit und ihres Nutzens für die Generierung lernförderlicher Rückmeldungen.

Das *Constraint-Based Modelling* (CBM) konzentriert sich primär auf die Identifikation von Regelverletzungen und liefert damit Hinweise auf typische Fehlkonzepte. Für differenziertere Rückmeldungen – etwa zu *semantisch korrekten, aber suboptimalen* Lösungen im Hinblick auf Effizienz, Verständlichkeit oder Generalisierbarkeit – bleibt der Ansatz jedoch unscharf. Insbesondere fehlt die Möglichkeit, innerhalb formal richtiger Lösungen unterschiedliche Bearbeitungsstrategien oder qualitative Unterschiede im Vorgehen abzubilden.

Auch datenbasierte Clustering-Verfahren stoßen an Grenzen: Zwar lassen sich damit empirisch Antwortmuster erkennen und gruppieren, doch beruhen die resultierenden Cluster meist auf

statistischen Ähnlichkeiten im Datenmaterial, nicht zwingend auf didaktisch relevanten Unterscheidungen. Dadurch besteht die Gefahr, dass zentrale Aspekte des Lernverhaltens oder bedeutsame konzeptuelle Differenzen unberücksichtigt bleiben. Für die Generierung erklärbarer und handlungsleitender Rückmeldungen ist dies jedoch entscheidend – wie auch die in Kapitel 4 dargestellten Analysen zur Feedbackpraxis von Lehrpersonen verdeutlichen.

Vor diesem Hintergrund verfolgt das Konzept der *Antwortklassen* einen hybriden Ansatz: Ziel ist es, typische Antwortmuster auf der Grundlage didaktisch motivierter Kriterien zu erfassen und diese mithilfe strukturierter Analysemethoden systematisch einzuordnen. Im Gegensatz zu rein regelbasierten Verfahren wie CBM oder rein datengetriebenen Clustering-Ansätzen erfolgt die Zuordnung von Antworten hier nicht allein aufgrund statistischer Ähnlichkeit, sondern auf Basis fachlich und didaktisch begründeter Kriterien. Die resultierenden Gruppen – die *Antwortklassen* – sind somit sowohl inhaltlich interpretierbar als auch technisch zuverlässig identifizierbar.

Diese Idee, Lernendenantworten systematisch zu typisieren, knüpft an Vorarbeiten zur Entwicklung von Fehlerklassifikationen an. Besonders hervorzuheben ist das von Zehetmeier et al. (2015) entwickelte Schema, das typische Fehler in der Programmierausbildung auf Basis der kognitiven Dimensionen der revidierten Bloom-Taxonomie systematisiert. Zehetmeier et al. (2015) identifizieren hierbei sieben Fehlerklassen – von *Mental Typo* (Flüchtigkeitsfehler) über *Knowledge Gap* und *Misconception* bis hin zu *Lack of Innovation*. Jede Klasse verweist dabei auf Defizite in spezifischen Basis-Kompetenzen und setzt somit voraus, dass die zugrunde liegenden kognitiven Prozesse zumindest teilweise rekonstruiert werden können.

Für die Analyse von Lernendenantworten in digitalen Lernumgebungen ist eine solche kognitive Rekonstruktion jedoch nur eingeschränkt möglich. Wie in Abschnitt 5.1.2 erläutert, lassen sich Denkprozesse nicht unmittelbar aus beobachtbaren Produkten ableiten, sondern nur indirekt erschließen. Selbst in der kognitiven Neurowissenschaft gilt es als problematisch, von beobachtbaren Daten direkt auf mentale Zustände zu schließen: So zeigt Poldrack (2006), dass selbst die Analyse von Gehirnaktivitäten keine verlässlichen Rückschlüsse auf zugrunde liegende kognitive Prozesse erlaubt. Vor diesem Hintergrund soll bei der Analyse von Programmcode oder anderen digitalen Artefakten im Rahmen dieser Arbeit nicht der kognitive Entstehungsprozess im Vordergrund stehen, sondern dessen *beobachtbares Resultat*. Begriffe wie *Fehlvorstellung* oder *Mental Typo* implizieren hingegen eine Interpretation der mentalen Zustände der Lernenden, die aus Code-Artefakten allein nicht valide abgeleitet werden kann.

Im Unterschied zu den auf kognitive Dimensionen ausgerichteten Fehlerklassen von Zehetmeier et al. (2015) sind *Antwortklassen* somit als objektiv beobachtbare, deskriptive Kategorien konzipiert. Sie erlauben eine eindeutige Zuordnung von Artefakten, ohne Annahmen über interne Denkprozesse treffen zu müssen. Dadurch wird eine systematische und reproduzierbare Klassifikation auch in automatisierten Analysesystemen möglich. Gleichzeitig erweitern *Antwortklassen* den diagnostischen Blick über Fehler hinaus: Sie erfassen auch korrekte, strategisch unterschiedliche und stilistisch variierende Lösungen und eröffnen so eine breitere Perspektive auf Lernendenverhalten. Als didaktisch-diagnostisches Instrument bilden sie damit eine Grundlage für kontextsensitive, lernförderliche Feedback-Strategien, die auf beobachtbare Muster statt auf hypothetische Kognitionen reagieren.

Auf Grundlage der Forschung zu Bewertungsinstrumenten in Form von Rubriken (Wolf und Stevens 2007; Reddy und Andrade 2010) lassen sich fünf zentrale Nutzenkategorien bestimmen, die deren Bedeutung für Unterricht, Prüfung und digitale Lernsysteme hervorheben:

ST – Save Time:
Die Klassifikation von Antworten reduziert individuellen Analyseaufwand. Diagnosen und Rückmeldungen lassen sich vorbereiten und automatisieren, wodurch sowohl Lehrende als auch Lernende profitieren.

BF – Better Feedback: Die systematische Zuordnung zu Antwortklassen ermöglicht gezielte, konsistente und qualitativ hochwertige Rückmeldungen, die über pauschale Hinweise hinausgehen und konkrete Lernimpulse geben.

MOC – More Objective Categorization: Die Orientierung an objektiv beobachtbaren Kriterien fördert transparente und vergleichbare Diagnosen.

BPE – Better Pre-Estimation: Die Vorabdefinition möglicher Antwortklassen sensibilisiert Lehrende bereits im Aufgabenentwurf für potenzielle Missverständnisse und trägt zur Validität von Aufgaben bei.

IIT – Improve Instructional Teaching: Die Aggregation und Analyse von Antwortklassen liefert empirisch fundierte Hinweise auf inhaltliche Schwächen, strategische Fehlkonzepte oder strukturelle Unklarheiten im Lehrangebot.

5.2.1 Definition und Grundidee von Antwortklassen

Das Konzept der *Antwortklassen* (AC) bildet den zentralen Bestandteil des hier vorgestellten Modellierungsansatzes für Lernendenantworten. Während sich frühere Arbeiten häufig auf die systematische Erfassung von Fehlern konzentrierten (vgl. Zehetmeier et al. (2015)), wird hier ein umfassenderes Konzept vorgeschlagen: Neben fehlerhaften Antworten werden auch korrekte Lösungen und deren zugrunde liegende Strategien systematisch erfasst. Damit erweitert das Konzept der Antwortklassen die Idee von Fehlerklassen um eine didaktisch breitere Perspektive, die nicht nur auf Defizite, sondern auch auf unterschiedliche Bearbeitungswege und Qualitätsebenen eingeht.

Ziel ist es, typische Antwortmuster strukturiert zu erfassen, objektiv zu beschreiben und zuverlässig zuordnen zu können. Formal lassen sich Antwortklassen als Mengen möglicher Antworten definieren, die bestimmte, klar beschriebene Merkmale erfüllen:

Antwortklasse

Sei R eine konkrete Antwort auf eine Aufgabe und B_x eine Beschreibung beobachtbarer Merkmale oder Strukturen. Dann gilt:

$$AC_x = \{R \mid R \text{ erfüllt alle Anforderungen der Beschreibung } B_x\}$$

Eine Antwort R gehört zur Antwortklasse AC_x genau dann, wenn sie sämtliche in B_x beschriebene Kriterien erfüllt.

Jede Antwortklasse besteht aus zwei zentralen Komponenten:

- einem **Identifikator** (AC_x), der die Klasse referenzierbar macht,
- und einer **Antwortklassenbeschreibung** B_x, die definiert, welche Kriterien erfüllt sein müssen, damit Antworten dieser Klasse zugeordnet werden.

Aus der Definition ergibt sich, dass eine Lernendenantwort mehreren Antwortklassen gleichzeitig zugeordnet werden kann, sofern sie die jeweiligen Kriterien erfüllt. Diese Mehrfachzugehörigkeit ist nicht nur zulässig, sondern aus didaktischer Perspektive ausdrücklich erwünscht, da Lernende häufig Lösungsansätze zeigen, die verschiedene Merkmale oder Strategien zugleich aufweisen

(vgl. Kapitel 4). Der Ansatz ermöglicht dadurch eine differenzierte Beschreibung komplexer Bearbeitungen und trägt dazu bei, unterschiedliche Aspekte einer Lösung systematisch miteinander in Beziehung zu setzen.

Die damit verbundene Offenheit erlaubt es, verschiedene Aspekte einer Antwort systematisch miteinander zu verknüpfen und überlappende Klassifikationen abzubilden. Dadurch entsteht ein anschlussfähiges Beschreibungsmodell, das sowohl für die theoretische Analyse als auch für algorithmische Verfahren in Lernsystemen anschlussfähig ist und vielfältige Kombinationen von Antwortmerkmalen zulässt.

Damit Antwortklassen sowohl für die didaktische Analyse durch Lehrpersonen als auch für die algorithmische Verarbeitung in digitalen Lernsystemen praktikabel sind, müssen die Klassifikationskriterien bestimmte Anforderungen erfüllen. Im Anschluss an die in Kapitel 4 dargestellten Befunde zur Feedbackpraxis gilt insbesondere: Nur wenn Rückmeldungen auf klar beobachtbaren Eigenschaften beruhen, können sie konsistent, nachvollziehbar und gerecht sein. Die konsequente Einhaltung der folgenden Anforderungen stellt sicher, dass Antwortklassen sowohl im Unterricht als auch in digitalen Lernumgebungen als belastbare *Grundlage* für Diagnose, Feedback und Steuerung dienen können. Zwei zentrale Anforderungen sind:

IF **Interpretationsfreiheit:** Die Klassenzugehörigkeit darf sich nicht auf mutmaßliche mentale Zustände oder subjektive Einschätzungen stützen. Aussagen wie "fehlendes Verständnis" oder "unsauber gearbeitet" sind ungeeignet, da sie Hypothesen über mentale Zustände der Lernenden darstellen, die sich aus einer gegebenen Antwort allein nicht belastbar ableiten lassen.

OB **Objektivität:** Die Kriterien müssen so formuliert sein, dass sie anhand der gegebenen Antwort zuverlässig festgestellt werden können. Das bedeutet: Zwei unabhängige Beurteilende sollten zur gleichen Einschätzung gelangen, und die Kriterien sollten soweit möglich auch algorithmisch überprüfbar sein.

Zwischen den Begriffen *objektiv beobachtbar* und *objektiv überprüfbar* besteht ein wichtiger Unterschied: *Beobachtbarkeit* bedeutet, dass eine Eigenschaft einer Antwort von verschiedenen Personen unabhängig voneinander konsistent erkannt werden kann. Sie stellt die didaktische Mindestanforderung dar, um Reliabilität und Transparenz zu gewährleisten. *Überprüfbarkeit* geht darüber hinaus und bezeichnet die Möglichkeit, die Klassenzugehörigkeit algorithmisch und reproduzierbar zu bestimmen. Damit eröffnet sich das Potenzial für Automatisierung, Skalierbarkeit und eine konsistente Nutzung in digitalen Lernsystemen. Während Beobachtbarkeit die pädagogische Grundlage für faire Diagnosen bildet, stellt Überprüfbarkeit das technische Ideal für eine vollständig automatisierte Klassifikation dar.

Zur Veranschaulichung des Konzepts der Antwortklassen wird das zuvor eingeführte *Running Example* (Abbildung 5.1) erneut aufgegriffen. Anhand dieser Programmieraufgabe wird exemplarisch gezeigt, wie typische Merkmale, Strategien und Fehlermuster in Lernendenantworten identifiziert und durch geeignete Antwortklassen systematisch beschrieben werden können. Die folgenden Beispiele verdeutlichen, dass Antwortklassen funktionale, strategische, stilistische und formale Aspekte gleichermaßen erfassen können – und somit eine Grundlage schaffen, um Rückmeldungen auf mehreren Ebenen differenziert zu gestalten.

Tabelle 5.2 zeigt exemplarische Antwortklassen für die Implementierung einer `minSearch`-Methode, wie sie in Abbildung 5.1 gefordert ist. Die Aufstellung versteht sich nicht als abschließende Taxonomie, sondern als Demonstration, wie sich durch die Kombination mehrerer Klassen komplexe Antwortmuster modellieren und diagnostisch auswerten lassen. Da die Aufgabenbeschreibung keine konkreten Einschränkungen bezüglich des Lösungsweges enthält, bieten verschiedene

Ansätze eine akzeptierten Soll-Zustand. Im Folgenden werden drei typische Strategien kontrastiert und jeweils den relevanten Antwortklassen (AC) zugeordnet:

ID	Beschreibung
AC_1	Die Datei `Homework3.java` lässt sich ohne Compilerfehler mit Java 24 übersetzen.
AC_2	Die Methode `minSearch` enthält mindestens einen Syntaxfehler.
AC_3	Die Methode `minSearch` liefert für gültige Eingaben das korrekte Ergebnis zurück.
AC_4	In der Methode `minSearch` wird eine bedingte Wiederholung verwendet.
AC_5	Ergebnis wird mit `System.out.println` ausgegeben, anstatt über `return` zurückgegeben zu werden.
AC_6	Die Implementierung verwendet Klassen oder Methoden der Java-Standardbibliothek `java.util`.
AC_7	Die Implementierung berücksichtigt den Spezialfall für leere Arrays.
AC_8	Die Methode `minSearch` enthält einen rekursiven Aufruf.
AC_9	Das Programm hat eine Laufzeitkomplexität von $\mathcal{O}(n)$.
AC_{10}	Konventionen zur Benennung von Variablen werden eingehalten (z. B. CamelCase).
AC_{11}	In der Klasse `Homework3.java` fehlt ein notwendiges Semikolon.
AC_{12}	Die Methode `minSearch` enthält Code, der nie ausgeführt wird (*Zombie-Code*).
AC_{13}	Die Methode `minSearch` hat den Rückgabetyp `void`.
AC_{14}	Die Implementierung verwendet `Arrays.sort` aus der Java-Standardbibliothek.

Tabelle 5.2: Beispiele für Antwortklassen für das Beispiel in Abbildung 5.1

Listing 5.1 zeigt eine Implementierung, bei der das übergebene Feld zunächst in aufsteigender Reihenfolge sortiert und dann das erste Element des sortierten Feldes zurückgegeben wird. Dieses Programm ist semantisch und syntaktisch korrekt – die Antwort liegt somit in AC_1 und AC_3. Da zum sortieren des Feldes die Java-API verwendet wurde, liegt die Antwort auch in AC_6 und damit insbesondere in AC_{14}.

```
import java.util.Arrays;
public class Homework3 {
    public int minSearch(int[] list) {
        Arrays.sort(list);
        return list[0];
    }
}
```

Listing 5.1: Implementierung mit API und Sortierung

Listing 5.2 zeigt eine Implementierung, die das Minimum des übergebenen Feldes direkt berechnet und zurückgibt – also ohne die Java-API auskommt. Diese Antwort kann damit den Antwortklassen AC_1, AC_3 und AC_4 zugeordnet werden, da eine bedingte Wiederholung verwendet wird, um die Einträge des Feldes einmal vollständig zu durchlaufen. Eine Variable `actualMin` wird verwendet, um den aktuell kleinsten gefundenen Wert des gegebenen Feldes zu speichern. Da der Bezeichner dieser Variable den Konventionen der Java-Programmiersprache entspricht und zusätzlich einen sprechenden Namen hat, kann die Antwort AC_{10} zugeordnet werden.

In der Implementierung in Listing 5.3 hingegen wird die Java-API verwendet um das Minimum direkt per Stream zu bestimmen. Sie gehört zu $AC_3 \cap AC_9$. Für ein leeres Feld würde `getAsInt()` jedoch eine Exception werfen. Insofern kann diese Implementierung auch AC_7 zugeordnet werden. Da das Programm jedoch aufgrund eines fehlenden Semikolons nicht kompiliert, gehört es auch zu AC_2 und AC_{11}.

```java
public class Homework {
    public int minSearch(int[] list) {
        int actualMin = list[0];
        for (int i = 1; i < list.length; i++){
            if (actualMin > list[i]) {
                actualMin = list[i];
            }
        }
    return min;
    }
}
```

Listing 5.2: Implementierung ohne API; iterative Suche

```java
import java.util.Arrays;
public class Homework {
    public int minSearch(int[] list) {
        return Arrays.stream(list).min().getAsInt()
    }
}
```

Listing 5.3: Implementierung mit API; direkte Berechnung

Die drei Beispiele illustrieren die *Nicht-Disjunktheit* von Antwortklassen: Sowohl Listing 5.2 als auch Listing 5.3 kann AC_9 zugeordnet werden. Die Implementierungen unterscheiden sich jedoch hinsichtlich der Nutzung der Java-API (AC_4 und AC_{14}). Das Programm in Listing 5.1 ist zwar semantisch korrekt, jedoch weniger effizient (*nicht* in AC_9) als Listing 5.2 und zudem potenziell mit Seiteneffekten verbunden (destruktive Änderung des Eingabearrays). Diese Differenzierungen sind didaktisch relevant, da sie unterschiedliche Zielsetzungen hinsichtlich einer Rückmeldung nahelegen (z. B. Effizienz, API-Nutzung, Umgang mit Randfällen) und damit zur Bereitstellung von qualitativ hochwertigem, konsistenten Feedback beitragen (vgl. **BF** in Abschnitt 5.2) sowie Kapitel 4. Auch wenn einige der in Tabelle 5.2 aufgeführten Antwortklassen auf den ersten Blick äquivalent erscheinen, gibt es zum Teil kleine Unterschiede in der Formulierung, die eine unterschiedliche Zuordnung ermöglichen. So gilt $AC_2 \subset \neg AC_1$ da AC_1 zum einen auf der gesamten Klasse `Homework3.java` basiert, während AC_2 nur die Methode `minSearch` betrachtet. Zum anderen zielt AC_2 auf einen Syntaxfehler ab, während AC_1 die generelle Kompilierbarkeit adressiert. Diese feinen Unterschiede in der Formulierung ermöglichen eine präzise und differenzierte Klassifikation, die sowohl für die didaktische Analyse als auch für die algorithmische Verarbeitung von Antworten relevant ist.

5.2.2 Ontologische Sichtweise als Strukturierungsperspektive

Das Konzept der Antwortklassen lässt sich auch in Form einer Ontologie modellieren, also als strukturierte und formale Repräsentation von Wissensdomänen, in der Klassen, Individuen, Eigenschaften und Relationen systematisch beschrieben werden. Diese Perspektive ergänzt die in Abschnitt 5.2.1 eingeführte mengentheoretische Definition um eine deklarative Grundlage, die insbesondere für die Erweiterung, Pflege und algorithmische Verarbeitung von Antwortmustern bedeutsam ist.

In der ontologischen Sichtweise repräsentieren Antwortklassen konzeptuelle Kategorien, denen konkrete Antworten als Instanzen zugeordnet werden. Die Zuordnung erfolgt auf Grundlage

definierter Eigenschaften (sog. *Properties*), die den Klassen zugrund liegen. Tabelle 5.3 zeigt die Entsprechung zentraler Begriffe zwischen Ontologie und Antwortklassensystem.

Ontologischer Begriff	Entsprechung im Antwortklassensystem
Klasse *concept/class*	Antwortklasse als Kategorie von Antworten
Individuum *individual*	konkrete Antwort als Instanz einer oder mehrerer Klassen
Eigenschaft *Property*	Beobachtbares Merkmal einer Antwort, das zur Klassendefinition genutzt wird (z. B. „*enthält return*“, „*nutzt for-loop*“)
Subklassen-Beziehung *Subclass-Relation*	Hierarchische Spezialisierung von Antwortklassen (z. B. `IterativeMinSuche` ist Subklasse von `StrategieIterativ`)

Tabelle 5.3: Ontologische Begriffe und ihre Entsprechung bei Antwortklassen

Die Modellierung von Antwortklassen als Ontologie eröffnet mehrere didaktische und technische Vorteile bei der Konzeption, Pflege und Anwendung:

- **Explizite Wissensrepräsentation:** Klassen und ihre Eigenschaften werden deklarativ beschrieben. Dies schafft Transparenz und Nachvollziehbarkeit, unabhängig von einer konkreten Implementation.
- **Hierarchiebildung und Vererbung:** Allgemeine Klassen können durch spezifische Subklassen verfeinert werden.
- **Mehrfachklassifikation:** Eine Antwort kann mehreren Klassen gleichzeitig angehören.Dies ermöglicht mehrdimensionale Rückmeldungen durch Kombination von Antwortklassen (z. B. *korrekte, aber ineffizient Lösung*).
- **Kompatibilität mit semantischen Technologien:** Antwortklassen lassen sich perspektivisch in Ontologiesprachen wie OWL überführen [2]. Dadurch wird eine Integration in semantische Lernsysteme oder Lernanalytik-Plattformen möglich.
- **Unterstützung für Inferenz und Diagnose:** Logische Schlussfolgerungen können genutzt werden, um zusätzliche Rückmeldungen abzuleiten. Gehört eine Antwort etwa zur Klasse AC_8, impliziert dies eine bestimmte Bearbeitungsstrategie, die für die Rückmeldung didaktisch relevant ist.

Die Klasse `StrategieIterativ` kann als Oberklasse von `IterativeMinSuche` modelliert werden, wobei letztere zusätzlich die Eigenschaft `BerechnetMinimum` erfüllt:

$$\texttt{IterativeMinSuche} \sqsubseteq \texttt{StrategieIterativ} \sqcap \texttt{BerechnetMinimum}$$

Eine konkrete Antwort `R` kann zusätzlich auch zur Klasse `StilkonventionErfüllt` gehören, was eine kombinierte Klassifikation erlaubt:

$$\texttt{Antwort}_x \in \texttt{IterativeMinSuche} \sqcap \texttt{StilkonventionErfüllt}$$

Die ontologische Modellierung von Antwortklassen erlaubt somit eine präzise, erweiterbare und interoperable Repräsentation didaktischer Kategorien. Sie schafft Anschluss an semantische Tech-

[2] siehe dazu z. B. https://www.w3.org/TR/owl2-primer

nologien und stärkt zugleich die didaktische Anschlussfähigkeit, indem Kategorien explizit, beobachtbar und kombinierbar bleiben. Damit bildet sie eine konzeptuelle Brücke zwischen der manuellen Analyse von Antworten durch Lehrpersonen (vgl. Kapitel 4) und der algorithmischen Verarbeitung in adaptiven Lernsystemen. Durch die deklarative, ontologische Spezifikation von Eigenschaften entsteht zudem eine transparente Zuordnungsgrundlage, die Interpretationsspielräume reduziert und Vergleiche über Aufgabenvarianten erleichtert (**MOC**). Gleichzeitig macht die explizite Modellierung implizite Erwartungen sichtbar und unterstützt damit die präventive Qualitätssicherung im Aufgabendesign (**BPE**).

5.2.3 Systematische Strukturierung von Antwortklassen

Die bisherigen Erläuterungen haben verdeutlicht, dass Antwortklassen weit über die Erfassung rein formaler oder syntaktischer Merkmale hinausgehen. Sie können unterschiedliche Dimensionen einer Antwort adressieren – von funktionaler Korrektheit über Bearbeitungsstrategien bis hin zu stilistischen oder didaktisch motivierten Kriterien. Um diese Vielfalt systematisch erfassen zu können, bietet sich eine taxonomische Gliederung an. Angestrebt wird eine Strukturierung von Antwortklassen, die ihre Generalisierbarkeit und Wiederverwendbarkeit sichert, ihre Einsatzbereiche präzise definiert und so eine konsistente Nutzung für Diagnose- und Feedbackzwecke ermöglicht.

5.2.3.1 Strukturierung nach Gültigkeitsbereich

Die Wiederverwendbarkeit von Antwortklassen hängt maßgeblich davon ab, inwieweit sich Aufgabenmerkmale und -anforderungen über verschiedene Varianten hinweg erhalten. Gerade bei Programmieraufgaben zeigt sich, dass viele Aufgaben keine Neuentwicklungen sind, sondern als Variationen bestehender Aufgaben zu verstehen sind – etwa mit verändertem Aufgabentext, neuen Beispielen oder angepasster Anwendungsdomäne, bei gleichbleibender Problemlogik.

Eine im Rahmen einer Arbeitsgruppe entwickelte Taxonomie zur Variabilisierung von Programmieraufgaben (Brocker et al. 2024) beschreibt genau diese Formen der Variation: Sie unterscheidet, *welche Elemente* einer Aufgabe verändert werden können (z. B. Aufgabentext, Eingabe-Ausgabe-Beispiele, Codefragmente) und *welchem Zweck* die Variabilisierung dient (z. B. Anpassung an Zielgruppen, Schwierigkeitsgrad oder Kontext). Dabei bleibt bei vielen Varianten der inhaltliche Kern der Aufgabe erhalten – etwa, wenn nur Domäne oder Benennung geändert werden, nicht jedoch die zugrunde liegende Aufgabenlogik oder die Art der erwarteten Lösung.

Für die Modellierung von Antwortklassen bedeutet dies: Je nach Grad der Variation können Aufgabenvarianten ganz oder teilweise dieselben konzeptuellen Anforderungen und Lösungsstrukturen aufweisen. Entsprechend lassen sich auch die zugehörigen Antwortklassen vollständig oder in Teilen übernehmen, sofern die grundlegende Aufgabenlogik erhalten bleibt. Die Einteilung nach Gültigkeitsbereichen trägt diesem Prinzip Rechnung, indem sie unterscheidet, auf welcher Abstraktionsebene eine Antwortklasse anwendbar ist – von allgemein- über domänen- bis hin zu aufgabenspezifischen Kategorien:

1. **Allgemeingültige Antwortklassen:** Gelten kontextübergreifend, unabhängig von Aufgabe, Thema oder Fachgebiet. Sie sind besonders für metakognitive oder diagnostische Rückmeldungen relevant – etwa zur Identifikation von Nichtbearbeitung oder Bearbeitungsabbruch.
   Beispiele: „*Antwort ist leer*“ oder „*Antwort wurde durchgestrichen*“.
2. **Domänenspezifische Antwortklassen:** Gelten innerhalb einer Fachdomäne, z. B. der Programmierung und stützen sich auf domänentypische Qualitäts- und Funktionskriterien.
   Beispiele: „*Programm kompiliert fehlerfrei*“ oder „*Algorithmus X in Methode Y ist korrekt*“.

3. **Aufgabentypspezifische Antwortklassen:** Beziehen sich auf Gruppen verwandter Aufgaben, etwa der Implementierung verschiedener Sortieralgorithmen oder Aufgaben, die einen rekursiven Aufruf fordern.
   Beispiele: „*Antwort sortiert in aufsteigender statt absteigender Reihenfolge*“, „*Basisfälle sind korrekt implementiert*“.

4. **Aufgabenspezifische Antwortklassen:** Gelten ausschließlich für eine konkrete Aufgabenstellung.
   Beispiele: „*Fehlender Basisfall für $n == 0$*“ oder „*Alle in der Aufgabenstellung geforderten Attribute der Klasse `Haus` wurden korrekt implementiert*“.

5.2.3.2 Strukturierung nach Inhalt

Neben dem Gültigkeitsbereich lassen sich Antwortklassen danach strukturieren, welche Dimensionen einer Lösung sie adressieren. Tabelle 5.4 gibt einen Überblick über zentrale Kategorien, wie sie sich im Kontext der in Abbildung 5.1 dargestellten Aufgabe ergeben können.

Kategorie	Beschreibung	Beispiele
Syntax	Adressiert korrekte Verwendung der Sprachelemente (z. B. Kompilierbarkeit, Syntax)	AC_1, AC_2, AC_{11}
Funktion/ Semantik	Adressiert funktionale Korrektheit und erwartetes Verhalten	AC_3, AC_{13}
Strategie	Adressiert gewählte Lösungsstrategien/Ansätze (z. B. rekursiv, iterativ, API-basiert).	AC_4, AC_6, AC_8, AC_{14}
Robustheit/ Fehlerbehandlung	Adressiert Behandlung von Sonderfällen oder Grenzfällen (z. B. leeres Feld, negative Zahlen)	AC_7
Stil/Konvention	Adressiert Code-Stil, Struktur und Einhaltung von Konventionen	AC_{10}, AC_{12}

Tabelle 5.4: Strukturierung von Antwortklassen nach Inhalt inkl. Beispielen

Die inhaltlichen Kategorien sind als illustrative Beispiele einer möglichen Strukturierung zu verstehen und erheben keinen Anspruch auf Vollständigkeit. Sie sollen exemplarisch zeigen, dass Antwortklassen über reine Fehlerklassifikationen hinausgehen, indem sie sowohl korrekte als auch inkorrekte Lösungen sowie strategische und qualitative Dimensionen erfassen können.

5.2.3.3 Strukturierung nach algorithmischer Entscheidbarkeit

Für die Umsetzung in digitalen Lernumgebungen – insbesondere zur automatisierten Bereitstellung kontextsensitiven Feedbacks – spielt neben der didaktischen Tragfähigkeit auch die *algorithmische Entscheidbarkeit* von Antwortklassen eine zentrale Rolle. Gemeint ist die Frage, ob und in welchem Umfang sich die Zugehörigkeit einer gegebenen Antwort zu einer Antwortklasse algorithmisch überprüfen lässt. Im Anschluss an grundlegende Konzepte der theoretischen Informatik lassen sich drei Kategorien unterscheiden:

1. **Entscheidbare Antwortklassen:** Die Klassenzugehörigkeit kann durch ein deterministisches Verfahren in endlicher Zeit zuverlässig geprüft werden. Typische Beispiele beruhen auf syntaktischen oder strukturell eindeutig überprüfbaren Kriterien wie AC_1, AC_2 oder AC_{10}.

2. **Semi-entscheidbare Antwortklassen:** Die Zugehörigkeit lässt sich algorithmisch *bestätigen*, jedoch nicht in allen Fällen *widerlegen*. Verfahren terminieren zuverlässig bei positiven Treffern, nicht aber zwingend bei Gegenbeispielen. Dies betrifft vor allem inhaltlich komplexe, dynamische oder strategische Eigenschaften. Ein Beispiel hierfür ist AC_3, da funktionale Korrektheit durch Testfälle teilweise geprüft werden kann, aber ein vollständiger Beweis i. A. nicht automatisierbar ist.

3. **Nicht entscheidbare Antwortklassen:** Eine algorithmische Bestimmung ist grundsätzlich nicht möglich, etwa bei komplexen semantischen Eigenschaften, allgemeinen Laufzeitanalysen oder intentionalen Zuschreibungen. Ein Beispiel hierfür ist AC_9, da die allgemeine Laufzeitanalyse über das Halteproblem hinausgeht.

In welche Kategorie eine Antwortklasse eingeordnet wird, hängt somit von ihrer genauen Formulierung ab. Ein anschauliches Beispiel bietet das Kriterium, ob in einer Methode Rekursion verwendet wird. Zunächst scheint diese Eigenschaft klar erkennbar zu sein, tatsächlich verändert sich ihre Zuordnung zu den oben genannten Kategorien jedoch je nach konkreter Formulierung:

Wird die Frage rein *syntaktisch* verstanden – also ob sich im statischen Aufrufgraphen der Methode ein Zyklus befindet, der auf einen direkten oder indirekten Selbstaufruf hinweist –, so lässt sich dies deterministisch und vollständig prüfen. Der Quelltext einer Methode bildet einen endlichen Graphen von Funktionsaufrufen, und Zyklen können zuverlässig erkannt werden. In dieser Formulierung zählt die Antwortklasse zu den **entscheidbaren**. Wird hingegen das konkrete *Laufzeitverhalten* betrachtet, verändert sich die Einstufung. Soll etwa geprüft werden, ob es *mindestens eine Eingabe* gibt, für die die Methode tatsächlich rekursiv aufgerufen wird, kann ein positives Beispiel durch Ausführen und Beobachten gefunden werden. Ein sicherer Nachweis, dass *keine* solche Eingabe existiert, ist jedoch nicht möglich, da Programme potentiell nicht terminieren. Die Frage reduziert sich damit auf das Halteproblem, und die Antwortklasse ist lediglich **semi-entscheidbar**. Schließlich wird die Eigenschaft **unentscheidbar**, wenn sie für *alle* möglichen Eingaben gefordert wird – etwa: „Die Methode ruft sich für jede Eingabe rekursiv auf." Ob ein beliebiges Programm für alle Eingaben garantiert in einen rekursiven Aufruf gelangt, lässt sich algorithmisch nicht bestimmen, da dies eine vollständige Laufzeitanalyse voraussetzen würde, die über das Halteproblem hinausgeht.

Dieses Beispiel zeigt, dass bereits kleine Unterschiede in der Formulierung – zwischen „tritt im Code auf", „tritt mindestens einmal zur Laufzeit auf" und „tritt für alle Eingaben auf" – die Einstufung einer Antwortklasse von *entscheidbar* über *semi-entscheidbar* bis hin zu *nicht entscheidbar* verschieben können. Dies verdeutlicht, dass sich nicht jede didaktisch sinnvolle Antwortklasse vollautomatisch identifizieren lässt. Die Kategorisierung ermöglicht es jedoch, den Automatisierungsgrad einzelner Antwortklassen explizit zu benennen und bei der Entwicklung von Rückmeldesystemen gezielt zwischen automatischer Erkennung, manueller Klassifikation oder hybriden Verfahren zu unterscheiden. Beschreibungen wie „unsicherer Lösungsweg", „Fehlvorstellung bei der Rekursion" oder „fehlendes Verständnis" mögen aus diagnostischer Sicht plausibel erscheinen, lassen sich jedoch nicht direkt aus einer Antwort ableiten. Sie setzen zusätzliche Hypothesen über das mentale Modell der Lernenden voraus und sind damit im Allgemeinen unentscheidbar. Solche Zuschreibungen liefern keine wohldefinierte, überprüfbare Beziehung zwischen Antwortstruktur und Klassenzugehörigkeit, sodass verschiedene Beurteilende zu unterschiedlichen Einschätzungen gelangen könnten – ein klarer Widerspruch zur angestrebten Reproduzierbarkeit und Transparenz. Damit werden sowohl Standardisierung und Fairness in der Anwendung unterstützt (**MOC**) als auch die Grundlage für effiziente, (teil-)automatisierte Analysepipelines gelegt (**ST**). Die in Abschnitt 5.2.1 formulierten Anforderungen, dass Kriterien von Antwortklassen *objektiv beobachtbar* und *interpretationsfrei* sein müssen, sind daher nicht nur aus didaktischer Perspektive zentral, sondern zugleich eine notwendige Voraussetzung für eine algorithmische Umsetzung.

5.2.4 Entwicklung von Antwortklassen

Antwortklassen verstehen sich als *deklarative Artefakte*, mit denen Zustände von Antworten (z. B. funktionale Korrektheit, Strategiewahl, Robustheit) *diagnostisch* erfasst und für *kontextsensitive Rückmeldungen* sowie adaptive Folgeentscheidungen genutzt werden können (vgl. Abschnitt 5.2.1 and Kapitel 4). Ihre Entwicklung stützt sich auf zweierlei Quellen: (a) *institutionalisiertes Wissen* aus Fach- und Fachdidaktik (Regeln, Konventionen, Fehlertypologien) sowie (b) *empirische Evidenz* aus realen Bearbeitungen. Im Folgenden werden beide Entwicklungszugänge skizziert und in ein iteratives Prozessmodell überführt.

5.2.4.1 Daten- und wissensbasierte Ansätze

Grundsätzlich lassen sich zwei komplementäre methodische Zugänge für die Entwicklung von Antwortklassen unterscheiden: *wissensbasierte* und *datenbasierte* Ansätze. Beide verfolgen das Ziel, typische Muster in Lernendenantworten so zu beschreiben, dass sie didaktisch bedeutsam, objektiv beobachtbar und technisch identifizierbar sind. Sie unterscheiden sich jedoch hinsichtlich ihrer methodischen Herangehensweise und epistemologischen Grundlage.

Wissensbasierte Ansätze entwickeln Antwortklassen, indem sie bestehendes Domänen- und Didaktikwissen systematisch in beobachtbare Kriterien übersetzen. Ausgangspunkt sind formale Sprachspezifikationen, anerkannte Stil- und Strukturkonventionen, etablierte Fehlertaxonomien sowie curriculare Vorgaben. So können etwa syntaktische Eigenschaften durch Analyse des abstrakten Syntaxbaums (AST) präzise überprüft werden, z. B. ob eine Methode einen rekursiven Funktionsaufruf enthält oder eine bedingte Wiederholung korrekt initialisiert ist. Ebenso lassen sich Konventionen wie der CamelCase-Stil für Variablennamen oder die Pflicht zur Methodendokumentation in Antwortklassen fassen. Auch fachdidaktisch beschriebene Fehlermuster – etwa die typische Off-by-One-Problematik oder die Verwechslung von Referenz- und Wertsemantik – können als wissensbasierte Klassen operationalisiert werden. Schließlich ermöglichen curriculare Vorgaben, z. B. ein Verbot bestimmter API-Funktionen oder die Forderung nach einer spezifischen algorithmischen Strategie, eine formale Abbildung in Antwortklassen.

Der zentrale Vorteil dieses Ansatzes liegt in seiner theoretischen Fundierung und Nachvollziehbarkeit. Wissensbasierte Klassen sind unabhängig von einem konkreten Datensatz formulierbar, klar begründet und gut dokumentierbar. Dadurch lassen sie sich leicht kommunizieren, überprüfen und auf unterschiedliche Aufgabenvarianten übertragen. Sie schaffen zudem eine solide Grundlage für faire und transparente Diagnose- und Feedbackprozesse, da ihre Kriterien in der Regel objektiv überprüfbar sind.

Allerdings stößt der Ansatz an Grenzen, wenn Lernende unvorhergesehene oder kreative Lösungen entwickeln, die von bestehenden Regeln nicht abgedeckt werden oder in der Literatur bislang nicht beschrieben sind. Auch kann deduktive Modellierung dazu führen, dass wichtige, empirisch häufig auftretende Bearbeitungsmuster zunächst unberücksichtigt bleiben. Wissensbasierte Verfahren bieten somit eine stabile Ausgangsbasis, benötigen aber meist eine Ergänzung durch datenbasierte Ansätze, um die tatsächliche Vielfalt realer Lernendenlösungen vollständig zu erfassen.

Datenbasierte Ansätze entwickeln Antwortklassen auf der Grundlage von Mustern in Korpora aus Lernendenantworten. Typischerweise werden große Mengen von Antworten (z. B. aus Prüfungen oder Übungs-Assessments) gesammelt, analysiert und nach inhaltlichen oder strukturellen Ähnlichkeiten gruppiert. Dies kann z. B. durch Methoden des unüberwachten Lernens wie *Clustering*, *Embedding*-basierte Ähnlichkeitsmaße oder visuelle Exploration geschehen.

Das zentrale Ziel besteht darin, *emergente Antwortmuster* zu identifizieren, also regelmäßig auftretende Bearbeitungsformen oder Fehler, die im Vorfeld nicht antizipiert wurden. In einem zwei-

ten Schritt werden diese Cluster manuell interpretiert, typisiert und mit objektiv beobachtbaren Eigenschaften verknüpft – daraus entstehen neue Antwortklassen.

Beispiele für datenbasierte Klassenentwicklungen:

- Ein Cluster von 27 Studierendenlösungen ($n = 300$) zur `minSearch`-Aufgabe (Abbildung 5.1) fällt dadurch auf, dass bei der Suche nach dem Minimum im Feld immer bei Index 1 begonnen wird und damit der Index 0 nicht berücksichtigt wird. Dies könnte ein Hinweis auf ein potenzielles Missverständnis im Umgang mit Feldern sein. Daraus entsteht die Antwortklasse: "Feldzugriff startet bei Index 1".
- Eine Häufung von Lösungen verwendet `Arrays.sort()`, obwohl in der Aufgabenstellung die Nutzung der Java-API explizit im Rahmen des Übungsbetriebes untersagt ist. Daraus wird die Antwortklasse "Java-API verwendet" abgeleitet.
- In einer Sammlung von 1000 Antworten zeigt sich ein signifikantes Subcluster mit einer syntaktisch korrekten, aber semantisch fehlerhaften Wiederholungsstruktur. Diese wird formalisiert zu: "Zählvariable wird nach der bedingten Wiederholung erneut verändert".

Ein zentrales Potenzial datenbasierter Verfahren liegt darin, aus realen Lernendendaten bislang unbekannte Muster zu erschließen und daraus systematisch neue Antwortklassen abzuleiten. So zeigt etwa Strickroth (2024), wie sich große Mengen von Live-Coding-Lösungen automatisiert nach syntaktischen und funktionalen Fehlern gruppieren lassen. Die daraus entstehenden Fehlercluster geben Lehrenden einen unmittelbaren Überblick über wiederkehrende Bearbeitungsmuster und ermöglichen es, häufige Probleme gezielt im Plenum zu thematisieren. Aufbauend auf ähnlichen Grundideen demonstrieren Zehetmeier et al. (2015), dass solche datengetrieben identifizierten Muster nicht nur für situatives Feedback nutzbar sind, sondern sich zu verallgemeinerbaren, didaktisch fundierten Klassifikationsschemata verdichten lassen. Gemeinsam verdeutlichen diese Ansätze, dass datenbasierte Verfahren nicht nur explorativ neue Antwortmuster sichtbar machen, sondern eine Brücke schlagen können von empirischen Beobachtungen hin zu robusten, wiederverwendbaren und didaktisch anschlussfähigen Antwortklassen. Datenbasierte Verfahren sind somit wertvoll für die empirische Absicherung, Erweiterung oder Korrektur bestehender (wissensbasierter) Antwortklassifikationen. Sie sind explorativ, kontextspezifisch und oft aufschlussreich für die tatsächliche Heterogenität der Lernprozesse.

Die beiden Ansätze zur Entwicklung von Antwortklassen stehen nicht in Konkurrenz, sondern ergänzen sich komplementär: Wissensbasierte Ansätze liefern *a-priori-Strukturen*, die theoriegeleitet und unabhängig vom konkreten Antwortmaterial entworfen werden können. Sie eignen sich besonders für curriculare Planung und systematische Rückmeldung. Datenbasierte Ansätze hingegen sind *a-posteriori* und erfassen reale Bearbeitungspraxen und Antwortvarianten. Sie eignen sich für die Erweiterung und Validierung bestehender Antwortklassen sowie zur Identifikation kontextabhängiger Besonderheiten. Idealerweise erfolgt die Entwicklung iterativ: Zunächst werden wissensbasierte Antwortklassen entworfen, dann empirisch mit Daten angereichert und schließlich wieder regelbasiert formalisiert. Dieses Wechselspiel ermöglicht eine theoretisch fundierte, aber empirisch sensibilisierte Modellierung.

Ob deduktiv-theoriegeleitet oder induktiv-datengetrieben – beide Ansätze zur Entwicklung von Antwortklassen tragen auf ihre Weise dazu bei, Lernendenantworten systematisch zu erfassen, zu klassifizieren und für Diagnose- sowie Feedbackprozesse nutzbar zu machen. Die Kombination verspricht besonders robuste, anschlussfähige und adaptive Modelle. Damit diese Verzahnung nicht zufällig, sondern methodisch kontrolliert erfolgt, braucht es einen klar strukturierten Entwicklungsprozess, der (1) fach- und fachdidaktisches Wissen in prüfbare Kriterien überführt, (2) reale Antwortdaten systematisch auswertet und (3) beide Perspektiven iterativ rückkoppelt.

Im Folgenden wird ein Prozessmodell vorgestellt, das diesen Weg von der theoriegeleiteten Modellierung über die datengestützte Validierung bis hin zur kontinuierlichen Verfeinerung von Antwortklassen beschreibt und als methodische Leitlinie für künftige Arbeiten dienen kann.

5.2.4.2 Prozessmodell zur empiriegestützten Entwicklung

Im Sinne des Design-Science Research (DSR) (siehe Abschnitt 3.3.1) lassen sich Antwortklassen als *digitale Artefakte* verstehen, die sowohl den *IST-Zustand* einer konkreten Lernendenantwort analysierbar machen als auch die Spezifikation eines *SOLL-Zustands* erlauben. Dadurch wird ein systematischer Vergleich möglich: Die Menge der tatsächlich erfüllten Antwortklassen einer Antwort kann den im Aufgabendesign erwarteten Antwortklassen gegenübergestellt werden. Abweichungen – etwa fehlende, überschüssige oder alternative Antwortklassen – werden damit transparent und diagnostisch nutzbar.

Antwortklassen folgen dem DSR-Paradigma dabei in zweierlei Hinsicht: Sie sind einerseits *Konstrukt-Artefakte*, die ein bislang unstrukturiertes Phänomen – die Vielfalt von Lernendenantworten – in eine erklärbare, kombinierbare Form bringen. Andererseits ermöglichen sie durch die explizite Abbildung von IST- und SOLL-Zuständen einen strukturierten Diagnose- und Vergleichsprozess, der sowohl in summativen als auch in formativen Kontexten adaptiv eingesetzt werden kann. Damit entsteht ein iterativ weiterentwickelbares Instrument, das zugleich auf theoretischer Rigorosität (z. B. Entscheidbarkeit, Objektivität) und auf praktischer Relevanz (Feedbackqualität, Effizienz, Aufgabenvalidierung) beruht.

Ausgehend von dieser DSR-Perspektive ergibt sich, dass die Entwicklung von Antwortklassen nicht als einmaliger Entwurf verstanden werden kann, sondern als zyklischer Prozess von Konstruktion, Evaluation und Iteration. Das nachfolgende Prozessmodell konkretisiert diesen Zyklus in fünf Schritten (vgl. Abbildung 5.5):

STEP 1 – Initiale Erstellung (Konstruktion): Ausgangspunkt ist die theorie- und wissensbasierte Formulierung erster Antwortklassen. Sie übersetzen fachdidaktisches Wissen, Fehlertypologien und curriculare Vorgaben in beobachtbare Kriterien und bilden damit die erste Artefakt-Instanz.

STEP 2 – Plausibilisierung/Operationalisierung (Rigor): Im nächsten Schritt wird geprüft, ob die Klassenbeschreibungen konsistent, interpretationsfrei und objektiv überprüfbar sind (vgl. Abschnitt 5.2.1). Dieser Schritt entspricht dem Anspruch von DSR, dass Artefakte auf einer soliden theoretischen und methodischen Grundlage stehen müssen und dient zudem der Sicherstellung von Validität und Entscheidbarkeit. Gegebenenfalls werden Formulierungen von Antwortklassenbeschreibungen geschärft oder Antwortklassen entfernt.

STEP 3 – Zuordnung (Evaluation im Feld): Lernendenantworten werden systematisch den bestehenden Klassen zugeordnet. Dies erlaubt die erste ergebnisoffene Erprobung des Artefakts in realen Kontexten und bildet die Grundlage für empirische Evidenz. Die Zuordnung kann dabei manuell oder (teil-)automatisiert erfolgen.

STEP 4 – Analyse/Reflexion (Wissensbeitrag): Die Ergebnisse der Zuordnung werden quantitativ und qualitativ ausgewertet. Typische Muster, Häufigkeiten oder Cluster geben Hinweise auf die Relevanz der Klassen und offenbaren zugleich neue Phänomene.

STEP 5 – Revision/Erweiterung: Auf Grundlage der Analyseergebnisse wird der Korpus an Antwortklassen überarbeitet. Neue Klassen können ergänzt, bestehende präzisiert oder zusammengeführt werden. Der überarbeitete Korpus bildet die Grundlage für künftige Anwendungen und Folgeanalysen. Dadurch wird der Zyklus erneut gestartet und das Artefakt iterativ weiterentwickelt.

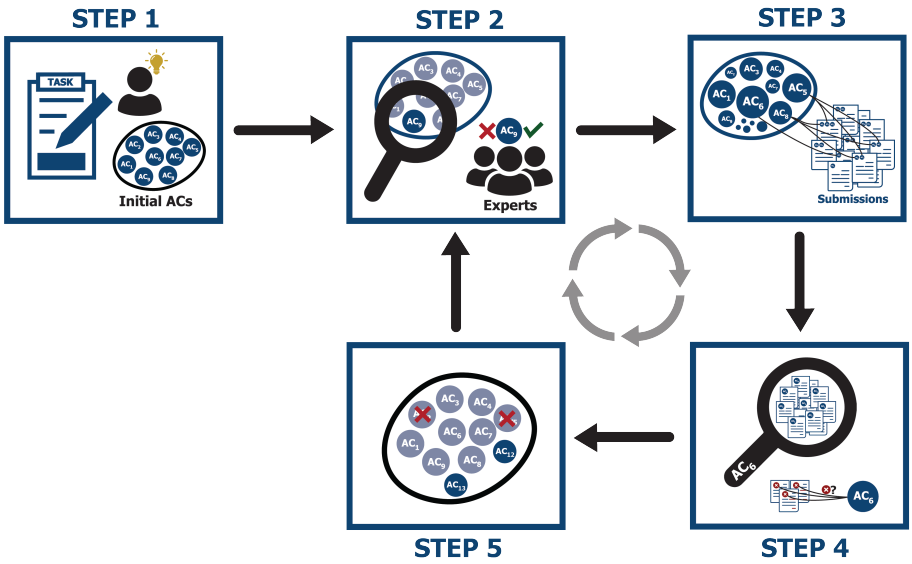


Abbildung 5.5: Iteratives Prozessmodell zur empiriegestützten Entwicklung von Antwortklassen

Das Prozessmodell operationalisiert damit den DSR-Grundgedanken: Artefakte entstehen nicht im einmaligen Entwurf, sondern werden in wiederholten Zyklen aus theoretisch fundierter Konstruktion und empirisch basierter Evaluation kontinuierlich verbessert. Auf diese Weise entwickeln sich Antwortklassen zu einem robusten, kontextspezifischen Instrument, das didaktische Tragfähigkeit (Beobachtbarkeit) und technische Umsetzbarkeit (Überprüfbarkeit) vereint.

5.2.5 Fallstudie: Anwendung des Konzepts in einer Klausur

Zur praktischen Erprobung des Konzepts der Antwortklassen wurde das gerade beschriebene iterative Prozessmodell in einer universitären Präsenzklausur der Lehrveranstaltung *Artificial Intelligence 1* eingesetzt. Ziel war es, erste Erfahrungen im Hinblick auf die in Abschnitt 5.2 skizzierten Potenziale zu sammeln und den Einsatz in einem realen Lehr-/Lernsetting exemplarisch zu untersuchen.

Insgesamt nahmen 462 Studierende an der 90-minütigen Klausur teil, die papierbasiert durchgeführt wurde. Der Großteil der Teilnehmenden war in den Masterprogrammen *Data Science* (n=225), *Artificial Intelligence* (n=118) und *Computer Science* (n=69) eingeschrieben. Die Klausur umfasste insgesamt 12 Aufgaben mit insgesamt 41 (Teil-)Aufgaben. Für sämtliche Aufgaben wurden die ersten drei Schritte des Prozessmodells (Initiale Erstellung, Plausibilisierung, Zuordnung) vollständig umgesetzt. Insgesamt wurden dabei vier domänenübergreifende sowie 192 aufgabenspezifische Antwortklassen entwickelt.

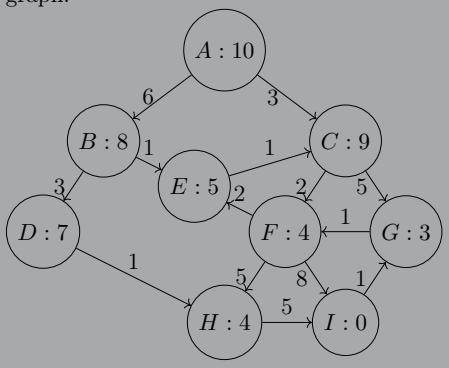
Da die Durchführung der Schritte 4 (Analyse) und 5 (Revision) für sämtliche 41 Teilaufgaben einen erheblichen personellen und zeitlichen Ressourceneinsatz erfordert hätte, welcher im Rahmen der Fallstudie nicht zur Verfügung stand, wurde für eine vertiefte Erprobung exemplarisch ein einzelner Aufgabenblock aus dem Gesamtkorpus ausgewählt.

Die Wahl fiel auf einen Block mit fünf Teilaufgaben zu verschiedenen Suchverfahren in einem gerichteten Graphen (u. a. A^*-, Greedy- und BFS-Suche). Dieser Aufgabentyp erwies sich als geeigneter Kompromiss zwischen offenen Programmieraufgaben mit großem Lösungsraum und stark geschlossenen Formaten wie Multiple-Choice: Zwar waren nur die jeweils besuchten Kno-

Task

Problem 2.1 (Search Algorithms) 8 pt

Consider the following graph:



Every node is labeled with $n : h(n)$ where n is the identifier of the node and $h(n)$ is the heuristic for estimating the cost from n to a goal node. Every edge is labeled with its actual cost.

Assume you have already expanded the node A. List the next four nodes (i.e., excluding A) that will be expanded using.

1. depth-first search 1 pt.
2. breadth-first search 1 pt.
3. uniform-cost search 2 pt.
4. greedy search 2 pt.
5. A*-search 2 pt.

If there is a tie, break it using alphabetical order.

Abbildung 5.6: Exemplarische Aufgabenstellung aus der Klausur *Artificial Intelligence 1*

ten anzugeben, sodass keine detaillierten Einblicke in individuelle Lösungswege möglich waren, dennoch erforderte die Bearbeitung die korrekte Anwendung algorithmischer Prinzipien auf eine konkrete Datenstruktur. Damit bot die Aufgabe genügend Komplexität, um typische Fehler- und Antwortmuster sichtbar zu machen, ohne den Rahmen einer ersten Erprobung durch zu hohe Varianz oder Interpretationsspielräume zu überlasten. Konkret sollten die Studierenden jeweils die nächsten vier Knoten benennen, die bei Anwendung des jeweiligen Algorithmus von einem bereits expandierten Knoten aus besucht würden (siehe Abbildung 5.6 für die vollständige Aufgabenbeschreibung).

Für jede (Teil-)aufgabe wurden zunächst von den Lehrperson initiale Antwortklassen formuliert (STEP 1). Anschließend wurde der erste Korpus an Antwortklassen im Plenum mit Kollegen aus der Fachdidaktik diskutiert und hinsichtlich ihrer Objektivierbarkeit und Abgrenzung überprüft (STEP 2). Die validierten Klassen wurden in ein Zuordnungsformular übertragen (siehe Abbildung 5.7), das die systematische Annotation sämtlicher Studierendenantworten mit den Antwortklassen ermöglichte (STEP 3). Die für die Korrektur verantwortlichen Personen erhielten darüber hinaus erläuternde Definitionen und Beispiele zu jeder Antwortklasse sowie Freifelder zur Erfassung bislang unberücksichtigter Phänomene.

Tabelle 5.5 zeigt den initialen Korpus an Antwortklassen für den ausgewählten Aufgabenblock. Die Klassen AC_1 bis AC_4 wurden als allgemeingültige Antwortklassen definiert und in allen Teilaufgaben verwendet (siehe Abschnitt 5.2.3.1). Da sich die Teilaufgaben nur durch den jeweils anzuwendenden Suchalgorithmus unterschieden, wurden die Klassen AC_5 bis AC_8 als Aufgabentypspezifische Antwortklassen deklariert.

Das zunächst entwickelte Antwortklassenkorpus war in seinem Zuschnitt stark auf Bewertung und Bepunktung ausgerichtet. Es erwies sich zwar als hilfreich für eine strukturierte Korrektur erwiesen sich jedoch für eine tiefergehende Analyse oder die Generierung **lernförderlicher**

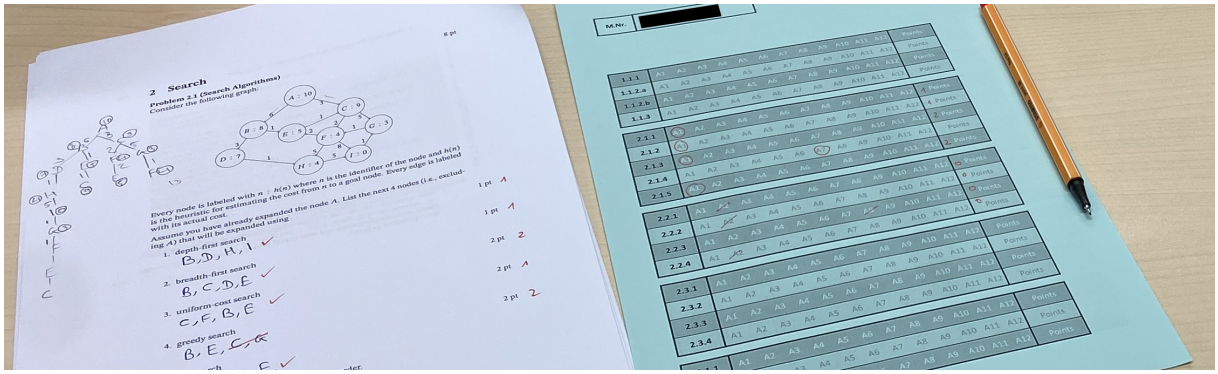


Abbildung 5.7: Manuelles Mapping der Antwortklassen auf Studierendenantworten

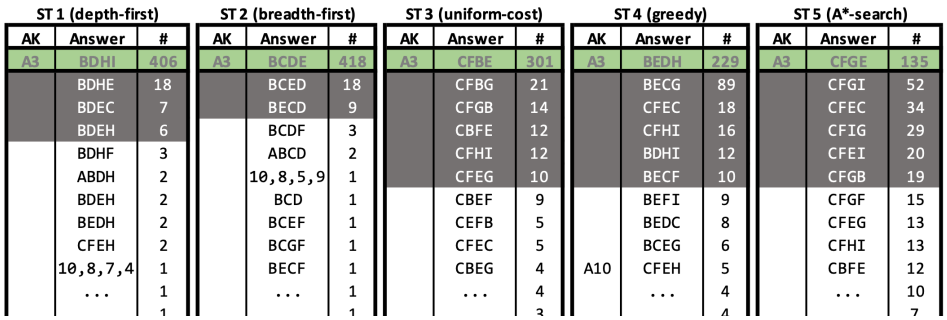
ID	Beschreibung
AC_1	Antwort ist leer.
AC_2	Antwort wurde durchgestrichen.
AC_3	Antwort ist vollständig korrekt.
AC_4	Antwort keiner existierenden AC zuzuordnen.
AC_5	Erster Knoten korrekt benannt.
AC_6	Zweiter Knoten korrekt benannt.
AC_7	Dritter Knoten korrekt benannt.
AC_8	Vierter Knoten korrekt benannt.

Tabelle 5.5: Initiale Antwortklassen für den Aufgabenblock *Search Algorithms*

Rückmeldungen als wenig aussagekräftig: Spezifische Fehlermuster, alternative Bearbeitungsstrategien oder konzeptionelle Besonderheiten blieben in dieser ersten Version weitgehend unberücksichtigt.

Zur differenzierten Weiterentwicklung des Korpus wurde eine Häufigkeitsanalyse aller annotierten Antworten durchgeführt (vgl. Abbildung 5.8). Die zugrunde liegende Annahme war dabei, dass (falsche) Antworten, die von einer großen Anzahl Studierender identisch gegeben wurden, ein identisches oder ähnliches Fehlermuster darstellen könnten und somit eine Rechtfertigung für weitere Untersuchungen darstellen. Neben der Gruppe der korrekten Antworten (erste Zeile, AC_3) zeigten sich mehrere große, wiederkehrende Antwortmuster. Für eine vertiefte Analyse wurden pro Teilaufgabe die fünf größten Fehlermustergruppen ausgewählt (Schwelle: mindestens fünf identische Antworten) und in einer Gruppensitzung mit erfahrenen Lehrpersonen diskutiert. Gruppen, für die *objektiv beobachtbare* Kriterien spezifiziert werden konnten, wurden als neue Antwortklassen aufgenommen (Auszug in Tabelle 5.6).

Beispiele: Häufig wurde der Startknoten fälschlich als erster Knoten der Sequenz genannt (AC_{10}), Knoten tauchten mehrfach in der Sequenz auf (AC_{11}), oder Gleichstände wurden entgegen der Vorgabe nicht alphabetisch aufgelöst (AC_{12}). Weitere Antwortklassen wie AC_{30} (Greedy-Suche: Auswahl lokaler Nachbarn statt global heuristisch günstigstem Knoten) wiesen auf algorithmische Fehlkonzeptionen hin. Ein besonders großes Cluster fand sich bei Teilaufgabe 4 zur Greedy-Suche: 89 von 462 Abgaben konnten AC_{30} (*lokale Nachbarschaftsauswahl*) zugeordnet werden. In der A^*-Teilaufgabe (Teilaufgabe 5) zeigte die größte Fehlermustergruppe (52 von 462) Anzeichen dafür, dass die Kantenrichtung zwischen G und I vertauscht wurde. Als wahrscheinliche Ursache identifizierten die Expertinnen und Experten die in der Aufgabenabbildung



ST 1 (depth-first)			ST 2 (breadth-first)			ST 3 (uniform-cost)			ST 4 (greedy)			ST 5 (A*-search)		
AK	Answer	#	AK	Answer	#	AK	Answer	#	AK	Answer	#	AK	Answer	#
A3	BDHI	406	A3	BCDE	418	A3	CFBE	301	A3	BEDH	229	A3	CFGE	135
	BDHE	18		BCED	18		CFBG	21		BECG	89		CFGI	52
	BDEC	7		BECD	9		CFGB	14		CFEC	18		CFEC	34
	BDEH	6		BCDF	3		CBFE	12		CFHI	16		CFIG	29
	BDHF	3		ABCD	2		CFHI	12		BDHI	12		CFEI	20
	ABDH	2		10,8,5,9	1		CFEG	10		BECF	10		CFGB	19
	BDEH	2		BCD	1		CBEF	9		BEFI	9		CFGF	15
	BEDH	2		BCEF	1		CEFB	5		BEDC	8		CFEG	13
	CFEH	2		BCGF	1		CFEC	5		BCEG	6		CFHI	13
	10,8,7,4	1		BECF	1		CBEG	4	A10	CFEH	5		CBFE	12
	...	1		...	1		...	4		...	4		...	10

Abbildung 5.8: Häufigkeitsverteilung gegebener Antworten im Aufgabenblock ST1-ST5

verwendeten, schwer erkennbaren Pfeilspitzen. Beide Befunde deuten somit auf *aufgabeninduzierte* Fehlerquellen (Darstellung, Instruktion) hin und nicht ausschließlich auf Defizite im Konzeptverständnis. Welche Missverständnisse oder Wissenslücken den jeweiligen Antwortklassen tatsächlich zugrunde liegen, lässt sich aus den Antworten allein nicht sicher bestimmen. Hierfür wären ergänzende Erhebungen wie Kurzinterviews mit den Studierenden notwendig. Gleichwohl erwiesen sich die neu identifizierten Antwortklassen als ergiebige Informationsquelle für weiterführende Analysen und die Ableitung gezielter Rückmeldungen.

Aufgabe	ID	Beschreibung
1-5	AC_{10}	Startknoten zu Beginn genannt.
1-5	AC_{11}	Knoten mehrfach in der Sequenz enthalten (Zyklen).
1-5	AC_{12}	umgekehrte alphabetische Reihenfolge, um Gleichstände aufzuheben.
1, 2	AC_{21}	Algorithmus auf Kosten (anstatt Heuristik) angewandt.
3, 4	AC_{22}	Algorithmus auf Heuristik (anstatt Kosten) angewandt.
2	AC_{31}	Sequenz entspricht den Knoten auf einer horizontalen Ebene des Graphen.
2	AC_{32}	Sequenz entspricht den Knoten auf einer vertikalen Linie des Graphen.
4	AC_{30}	Greedy-Suche auf Nachbarn des vorherigen Knotens beschränkt
5	AC_{30}	A*-Suche mit vertauschter Kantenrichtung bei $G \rightarrow I$

Tabelle 5.6: Beispiele neu identifizierter Antwortklassen

Die Umsetzung der in Abschnitt 5.2.4.2 beschriebenen Schritte war zunächst mit einem deutlichen initialen Mehraufwand verbunden. Ob sich dieser Aufwand in der Summe durch eine Zeitersparnis bei der Korrektur ausgleicht (vgl. **ST** in Abschnitt 5.2), lässt sich auf Basis der vorliegenden Fallstudie nicht abschließend bewerten. Perspektivisch erscheint eine Effizienzsteigerung jedoch plausibel – insbesondere bei standardisierten oder wiederkehrenden Aufgabenformaten sowie in Prüfungen mit geteiltem Korrekturaufwand. Auch bei Aufgabenvarianten, etwa durch leichte Kontextänderungen (z. B. Austausch des zugrunde liegenden Graphen), können bestehende Antwortklassen konzeptuell wiederverwendet und angepasst werden.

Besonders deutlich zeigte sich der Mehrwert des Ansatzes im Hinblick auf die Qualität und Konsistenz des Feedbacks (**BF**). Durch die systematische Verknüpfung von Antwortklassen mit vorformulierten Rückmeldungen lässt sich in groß angelegten Prüfungen einheitliches, nachvollziehbares und fachlich fundiertes Feedback skalierbar bereitstellen – ohne dass jede Lösung individuell kommentiert werden muss. Damit wird eine zentrale Herausforderung in der Programmierausbildung adressiert: die Bereitstellung von qualitativ hochwertigem Feedback unter realistischen Ressourcenkonditionen. Selbst eine partielle Abdeckung, etwa durch Rückmeldungen

zu den häufigsten Antwortklassen, kann die Feedbackqualität und Transparenz bereits spürbar erhöhen. Darüber hinaus zeigte sich, dass viele Antwortklassen über die konkrete Aufgabe hinaus generalisierbar sind – etwa als wiederkehrende Fehlermuster bei der Anwendung von Suchverfahren auf Graphen. Dieses Potenzial eröffnet neue Perspektiven für systematisches, über Aufgaben hinweg konsistentes Feedback und für die kontinuierliche Weiterentwicklung von Lernmaterialien. Antwortklassen fungieren damit zugleich als Instrument der Lernendenunterstützung und der Lehrdiagnostik.

Ein weiterer Befund betrifft den Zusammenhang zwischen Antwortklassen und typischen Bearbeitungsmustern. So deuteten zahlreiche Einordnungen in AC_{30} (Greedy-Suche) auf ein Vorgehen hin, bei dem stets der Nachbar des zuletzt besuchten Knotens gewählt wurde – entgegen der intendierten Strategie, den global heuristisch günstigsten Knoten zu bestimmen. Die Diskussion im Lehrteam ergab, dass ein in der Vorlesung verwendetes Beispiel eines Städtenetzes dieses Verhalten möglicherweise ungewollt nahegelegt hatte. Damit lieferte die Analyse der Antwortklassen einen konkreten Hinweis zur Überarbeitung der Aufgabenstellung und zur gezielten Thematisierung dieses potenziellen Missverständnisses in der Lehrpraxis (**IIT**). Ähnlich zeigte sich in der A*-Teilaufgabe eine auffällige Häufung der Antwortklasse (AC_{30}), die auf eine fehlerhafte Interpretation der Kantenrichtung hindeutete. Als mögliche Ursache wurde eine grafische Unschärfe in einer Abbildung identifiziert, bei der die Pfeilspitzen der Kanten schwer erkennbar waren. Beide Beobachtungen führten zu konkreten Anpassungen der Aufgabe und zeigen, dass Antwortklassen auch als Instrument der Aufgabenanalyse und Qualitätskontrolle nutzbar sind.

Bezüglich einer objektiveren Korrektur (**MOC**) zeigte sich ebenfalls ein Vorteil des Ansatzes: Explizit definierte, beobachtbare Kriterien reduzierten Interpretationsspielräume und erhöhten die Konsistenz bei verteiltem Korrekturaufwand. Werden Antwortklassen mit Punktwerten verknüpft, kann eine transparente und nachvollziehbare Bewertung auf Basis von Antwortklassen erfolgen. Ein Beispiel hierfür ist die Klasse AC_{10}: Während zuvor ein falscher Startknoten häufig zum vollständigen Punktabzug führte, ermöglichte die differenzierte Modellierung eine Teilanrechnung, ohne korrekte Folgeschritte vollständig zu entwerten.

Insgesamt bestätigt die Fallstudie die zentralen Potenziale des Antwortklassen-Konzepts: Es unterstützt Lehrpersonen bei der Analyse von Antworten, verbessert die Qualität und Einheitlichkeit der Analyse-Ergebnisse und fördert zugleich die Reflexion und Weiterentwicklung der Aufgabenstellung sowie damit verbundener Lehrmaterialien. Bemerkenswert ist dabei, dass sich diese Effekte nicht nur auf die Analyse individueller Antworten beschränken, sondern auch auf übergeordnete Aspekte wie Fairness, Aufgabenqualität und didaktische Kohärenz auswirken.

Darüber hinaus initiierte der Einsatz von Antwortklassen Reflexionsprozesse innerhalb des Lehrteams. Ein beteiligter Dozent fasste seine Einschätzung rückblickend wie folgt zusammen:

> *I was very skeptical whether this concept from secondary education could carry over to the university setting and whether the effort of AC annotation would be sustainable in a correction process that keeps my entire research group busy full-time for a week. Frankly, I only agreed to this to make my didactics colleague happy. [...] But the result of the experiment has utterly convinced me of the approach, even in a paper-based setting. Seeing the discussions among the graders induced by developing and annotating ACs and the content awareness in the grading process made the added effort worthwhile[...], and that is before we have taken a closer look at the data that this experiment generated.*

Die Rückmeldung verdeutlicht, dass das Konzept der Antwortklassen nicht nur zu effizienteren und konsistenteren Rückmeldeprozessen beiträgt, sondern auch eine vertiefte fachlich-didaktische Auseinandersetzung innerhalb von Lehrteams anregen kann – ein Effekt, der über die unmittelbare Prüfungssituation hinausweist.

5.3 Das Y-Modell zur Aufgaben- und Antwortformalisierung

Die bisherigen Analysen haben verdeutlicht, dass Aufgaben und Antworten aus mehreren Dimensionen bestehen, die zwar einzeln beschrieben werden können, deren didaktische Relevanz sich jedoch erst im Zusammenspiel entfaltet. So reicht es bspw. nicht aus, ausschließlich die in einer Aufgabenstellung genannten fachlichen Konzepte zu erfassen, da häufig implizite Voraussetzungen bestehen, die für die Bearbeitung zentral sind. Ebenso geben sprachliche Operatoren wie *implementieren* oder *analysieren* zwar erste Hinweise auf kognitive Anforderungen, lassen jedoch ohne zusätzliche Kontextinformationen keine verlässlichen Rückschlüsse auf das Anspruchsniveau zu. Schließlich entfaltet auch die Analyse von Lernendenantworten ihr volles diagnostisches Potenzial erst dann, wenn sie über dichotome Kategorien wie *richtig* und *falsch* hinausgeht und mit typischen Bearbeitungsmustern, Strategien oder Fehlkonzepten in Verbindung gebracht wird. Damit wird deutlich, dass die zuvor diskutierten Komponenten – *Wissen*, *kognitive Prozesse*, *Aufgabenstellung* und *Antworten* – nicht isoliert nebeneinanderstehen dürfen, sondern in ein integratives Modell überführt werden müssen. Im Rahmen dieses Kapitels werden die verschiedenen Dimensionen systematisch in einem Modell zusammengeführt (**FF2.3**) und so eine Grundlage für algorithmische Diagnose und Steuerung adaptiver Feedback-Strategien geschaffen.

In den folgenden Abschnitten wird mit dem **Y-Modell** ein solcher konzeptueller Ordnungsrahmen vorgestellt. Er integriert die zuvor entwickelten Ergebnisse in eine formalisierte Aufgaben- und Antwortbeschreibung und macht diese für adaptive Lernsysteme nutzbar. Ziel ist dabei nicht die Einführung einer weiteren isolierten Klassifikation, sondern die Entwicklung eines flexiblen, annotierbaren Modells, das unterschiedliche Perspektiven – fachlich, kognitiv, didaktisch und diagnostisch – miteinander verbindet. Das Y-Modell versteht sich damit nicht als Konkurrenz zu bestehenden Modellen, sondern als verbindender Bezugsrahmen und *konzeptuelles Meta-Modell* für die Modellierung, Analyse und automatisierte Verarbeitung von Aufgaben und Antworten in adaptiven, feedbackorientierten Lernsystemen. Im Fokus steht die Beschreibung der strukturierten Beziehungen zwischen den zuvor identifizierten Dimensionen. Zugleich legt das Modell fest, welche Elemente einer Aufgabe und ihrer Bearbeitung annotiert werden sollten, um sie maschinenlesbar, didaktisch interpretierbar und für algorithmische Verfahren in adaptiven Lernsystemen nutzbar zu machen. Eine grundlegende Annahme hierfür ist, dass Lernaufgaben stets als Einheit aus Aufgabenstellung, Wissensbezug, kognitiver Anforderung und möglichen Antwortformen verstanden werden. Erst durch ihre explizite Modellierung und Verknüpfung wird es möglich, differenzierte Feedback-Strategien systematisch abzuleiten und die Forschungsdesiderata dieser Arbeit einzulösen (vgl. Abschnitt 3.1).

5.3.1 Grundstruktur des Y-Modells

Die in Abschnitt 5.1 zentralen Komponenten von Aufgaben werden im sogenannten *Y-Modell* zu einer integrierten Struktur zusammengeführt. Es umfasst damit vier miteinander verknüpfte Dimensionen, mit denen zentrale Aspekte von Programmieraufgaben und Lernendenantworten systematisch beschrieben werden sollen:

- **Wissensdimension:** Welche fachlichen Inhalte oder Wissensbereiche sind für das Lösen der Aufgabe relevant?
- **Kognitive Dimension:** Welche kognitiven Prozesse werden von den Lernenden erwartet (z. B. identifizieren, vergleichen, begründen)?
- **Aufgabenstellung (Task):** Wie wird die Aufgabe konkret formuliert, dargestellt oder vermittelt?
- **Antwortdimension:** Welche Formen des Antwortverhaltens sind möglich, erwartbar oder gültig und wie können sie systematisch klassifiziert werden?

Diese werden im Modell in einer Y-förmigen Struktur angeordnet (siehe Abbildung 5.9). Die zentrale Komponente des Modells ist die Aufgabe selbst, die durch ihre konkrete *Aufgabenstellung* (Task) – also ihre sprachlich-formale Beschreibung und Darstellung – zum Ausdruck kommt (siehe Abschnitt 5.1.3). In dieser Aufgabenstellung manifestieren sich sowohl inhaltliche Anforderungen als auch strukturelle Merkmale – etwa verwendete Operatoren, Hinweisstrukturen, Darstellungsformate sowie Erwartungen an Interaktionen, Eingabeformate oder Bearbeitungsmodalitäten. Sie bildet damit den Bezugspunkt für alle weiteren konzeptuellen Elemente des Modells und ist somit nicht nur bloß Auslöser für Lernverhalten, sondern integraler Bestandteil der didaktischen Struktur einer Aufgabe.

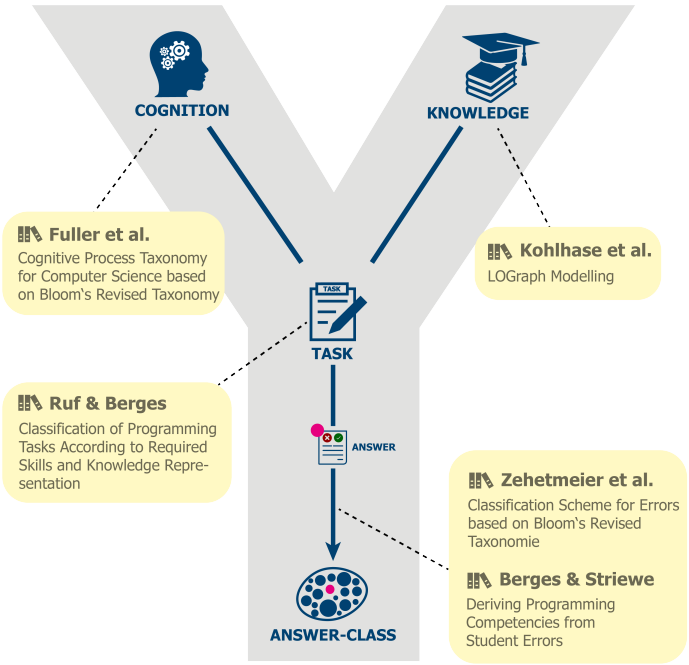


Abbildung 5.9: Das Y-Modell

Der obere Teil des Y-Modells veranschaulicht das Zusammenspiel zweier grundlegender Merkmale: der *Wissensdimension* und der *kognitiven Dimension*. In ihrer Kombination wird bestimmt, *was* gelernt, geübt oder geprüft werden soll (Wissen) und *wie* dieses Wissen verarbeitet, angewendet oder transformiert werden soll (kognitive Prozesse). Diese beiden Aspekte stellen die didaktische Grundlage der Aufgabenstellung dar – sie sind zwar nicht immer explizit formuliert, aber im Design jeder Lernaufgabe implizit angelegt.

Der untere Teil des Modells repräsentiert die *Antwortdimension*. Sie umfasst die erwartbaren Antwortklassen, die sich aus der spezifischen Kombination von Wissen und kognitiven Prozesse im Rahmen einer vorliegenden Aufgabenstellung ergeben können. Dabei handelt es sich nicht ausschließlich um typische Fehlermuster, sondern grundsätzlich um jede Form didaktisch relevanter Antwortkategorien – etwa funktionale, strategische, stilistische oder strukturell-konzeptionelle Merkmale, wie sie im Unterricht oder in Prüfungssituationen wiederholt beobachtbar sind (vgl. Abschnitt 5.2). Aufgaben und Antworten sind im Y-Modell somit konzeptionell untrennbar miteinander verbunden: Eine systematische Aufgabenformalisierung erfordert stets auch die strukturierte Modellierung der möglichen Reaktionsformen seitens der Lernenden.

5.3.2 Formalisierung der Wissensdimension

Die *Wissensdimension* bildet im Y-Modell die inhaltliche Achse von Lernaufgaben. Sie beschreibt, welche fachlichen Konzepte mit einer Aufgabe verbunden sind und in welchem Verhältnis diese zueinanderstehen. Grundlage dafür sind die in Abschnitt 5.1.1.2 eingeführten *Knowledge Components*, die als kleinste bedeutungstragende Wissenseinheiten modelliert werden. Jedes adressierte Konzept kann dabei entweder als Voraussetzung (`precondition`) oder als Ziel (`objective`) annotiert werden.

Die Wissensdimension macht somit explizit, welche inhaltlichen Bausteine durch eine Aufgabe aktiviert, vorausgesetzt oder aufgebaut werden. Sie stellt den inhaltlichen Bezugspunkt für die kognitive Dimension, die Aufgabenstellung und die Antwortdimension dar und verankert die Bearbeitung einer Aufgabe im fachlich-konzeptionellen Kontext. Damit schafft sie die Voraussetzung, um Lernprozesse nicht nur aufgabenbezogen, sondern entlang fachlicher Wissensstrukturen zu analysieren und zu steuern.

Zur formalen Repräsentation dient ein fachspezifischer *Wissensgraph*, in dem Konzepte als Knoten und deren Relationen als Kanten modelliert sind. Typische Relationen wie `erfordert`, `baut-auf` oder `Teil-von` ermöglichen es, die konzeptuelle Einbettung einer Aufgabe transparent zu machen und Abhängigkeiten zwischen Wissenselementen für Diagnose- und Feedbackprozesse nutzbar zu machen. Wissensgraphen eignen sich in besonderer Weise für diese Form der Repräsentation, da sie nicht nur einzelne Konzepte isoliert beschreiben, sondern auch deren konzeptuelle Einbettung und semantische Vernetzung abbilden können. Dies ermöglicht es, Aufgaben systematisch entlang konzeptueller Pfade zu analysieren, aufeinander aufzubauen oder adaptive Rückmeldungen mit Bezug auf nicht erfüllte Voraussetzungen zu generieren.

Technische Anforderungen

Damit die Wissensdimension operabel wird, sind insbesondere folgende Voraussetzungen für eine technische Umsetzung zu erfüllen:

- W1 ein zentrales, durchsuchbares Repository für fachliche Konzepte und deren Abhängigkeiten in Form eines Wissensgraphen,
- W2 standardisierte Schnittstellen zur Annotation von Aufgaben mit referenzierten Konzepten, inklusive Annotation der Rolle (z. B. `precondition`- und `objective`-Markierungen),
- W3 Werkzeuge zur konsistenten Erstellung, Pflege und Erweiterung des Wissensgraphen (z. B. durch Autorensysteme oder domänenspezifische Editoren),
- W4 Mechanismen zur automatisierten oder assistierten Analyse und Validierung von Konzeptverknüpfungen

5.3.3 Formalisierung der kognitiven Dimension

Die *kognitive Dimension* verankert im Y-Modell die mit der Aufgabe verbundenen Denk- und Verarbeitungsprozesse. Sie ergänzt die inhaltliche Verortung der *Wissensdimension* um die Frage, *wie* Lernende mit den adressierten Konzepten umgehen sollen – etwa durch Reproduktion, Anwendung, Analyse oder kreative Transformation (vgl. Abschnitt 5.1.2). Zur Modellierung dieser Prozesse kann prinzipiell jede etablierte Lernzieltaxonomie herangezogen werden, die kognitive Anforderungen in unterschiedlichen Abstraktions- oder Komplexitätsstufen beschreibt. Allgemein erscheint jedoch eine domänenspezifische Differenzierung als besonders zweckmäßig, da dadurch charakteristischen Lernaktivitäten der jeweiligen Disziplin – etwa das Verständnis und die Produktion von Code bei der Programmierung – explizit abgebildet werden können.

Aus diesem Grund wird im vorliegenden Modell die für die Informatik adaptierte Matrix nach Fuller et al. (2007) herangezogen. Sie operationalisiert kognitive Anforderungen, indem sie *rezeptive* Prozesse (z. B. Lesen, Verstehen, Erkennen) und *produktive* Prozesse (z. B. Entwickeln, Entwerfen, Implementieren) als orthogonale Dimensionen modelliert (vgl. Abschnitt 5.1.2.2). Diese Unterscheidung erlaubt es, Aufgaben präziser in Bezug auf die Art der geforderten kognitiven Aktivität zu verorten und trägt damit zu einer kontextspezifischeren Beschreibung programmierbezogener Lernprozesse bei.

Im Y-Modell werden kognitive Anforderungen als strukturierte Annotationen geführt. Sie beziehen sich auf die *intendierte* kognitive Operation der Aufgabenstellung, sind auf der Ebene der referenzierten *Knowledge Components* (KnoCs) verankert und können mehrwertig sein, indem sie rezeptive und produktive Aspekte miteinander kombinieren. Für jedes betroffene Konzept werden eine oder mehrere kognitive Kategorien vergeben, sodass ein präzises Anforderungsprofil entsteht, das in Diagnose-, Auswahl- und Feedbackentscheidungen eingebunden werden kann (vgl. Abschnitt 5.3).

Die in Abbildung 5.1 dargestellte Beispielaufgabe zur Bestimmung des Minimums eines Feldes mit Ganzzahlen erfordert mehrere aufeinander bezogene kognitive Aktivitäten, die sich entlang der beiden Achsen der Fuller-Taxonomie differenzieren lassen. Naheliegend ist zunächst, dass Lernende das mathematische Konzept eines Minimums in seiner Bedeutung erfassen und verstehen, bevor sie dieses Wissen durch die Anwendung geeigneter Kontrollstrukturen in eine algorithmische Lösung übersetzen. Für fortgeschrittene Lernende kann die Aufgabe darüber hinaus die Anforderung enthalten, alternative Strategien zu entwickeln und kritisch zu bewerten.

Um diese Anforderungen präzise abzubilden, wird im Y-Modell eine standardisierte Syntax für kognitive Annotationen verwendet: `{{Producing} X {Interpreting}; KnoC}`. Das Kreuzprodukt von `Producing`- und `Interpreting`-Kategorien entspricht dabei den Dimensionen der Taxonomie nach (Fuller et al. 2007) und wird mit dem jeweils betroffenen fachlichen Konzept aus dem Wissensgraphen (KnoC) kombiniert. Auf diese Weise entsteht eine mehrdimensionale Beschreibung, die sowohl intendierte kognitive Prozesse als auch deren inhaltlichen Bezug sichtbar macht.

Für die `minSearch`-Aufgabe in Abbildung 5.1 ergeben sich folgende beispielhafte Annotationen:

- `{NONE, UNDERSTAND; minimum}`
  Das Konzept des Minimums muss inhaltlich erfasst und verstanden werden, ohne dass dabei ein neues Produkt erzeugt wird. Es handelt sich also um eine rein rezeptive Anforderung.
- `{APPLY, REMEMBER; for-loop}`
  Die Aufgabe erfordert das aktive Anwenden einer Wiederholung mit fester Anzahl (z. B. `for`) auf ein neues Problem. Gleichzeitig wird deklaratives Wissen über die Syntax und Funktionsweise dieser Wiederholungsstruktur vorausgesetzt und muss erinnert werden.
- `{APPLY, UNDERSTAND; comparison}`
  Die Vergleichslogik innerhalb der Schleife muss korrekt umgesetzt werden. Dies setzt voraus, dass Lernende die Bedeutung relationaler Operatoren verstehen und dieses Verständnis zielgerichtet in der Implementierung anwenden.

Technische Anforderungen

Damit die kognitive Dimension im Sinne des Y-Modells systematisch genutzt werden kann, sind folgende Anforderungen einer technische Umsetzung zu erfüllen:

K1 Ein definierter Katalog kognitiver Kategorien, basierend auf einer geeigneten Taxonomie mit klarer Semantik.

K2 Unterstützung mehrwertiger, konzeptbezogener Annotationen (Bindings *KnoC* ↔ *kognitive Kategorie(n)*),

K3 Werkzeuge zur Annotation und Visualisierung kognitiver Anforderungen.

K4 Schnittstellen zur Weiterverwendung in Diagnose, adaptiver Aufgabenauswahl und Feedbacklogiken.

5.3.4 Formalisierung der Aufgabenstellung

Die *Aufgabenstellung* bildet im Y-Modell die kommunikative Schnittstelle zwischen Aufgabe und Lernenden. In ihr verdichten sich die inhaltlichen Anforderungen der Wissensdimension und die prozessualen Anforderungen der kognitiven Dimension zu einem sprachlich-strukturierten Auftrag, der den Bearbeitungsprozess steuert und zugleich das Antwortverhalten rahmt. Damit fungiert sie als zentrales Bindeglied: Sie übersetzt didaktische Intentionen in eine für Lernende interpretierbare Form und schafft zugleich die Grundlage für algorithmische Analyse sowie adaptive Feedbacksteuerung. Die formalisierte Modellierung der Aufgabenstellung trägt somit dazu bei, die verschiedenen Dimensionen des Y-Modells zu einer operablen Gesamtrepräsentation zusammenzuführen (siehe **FF2.3**, Abschnitt 3.2).

Aufbauend auf den in Abschnitt 5.1.3 beschriebenen zentralen Komponenten der Aufgabenstellung werden im Y-Modell vier annotierbare Elemente unterschieden, die jeweils spezifische didaktische Funktionen erfüllen und zugleich für die maschinelle Verarbeitung relevant sind:

- **Operator:** zentrale Handlungsbegriffe wie *implementieren*, *analysieren* oder *beschreiben*, die kognitive Prozesse anstoßen und als Marker für die kognitive Dimension fungieren
- **Aufgabenformat:** die erwartete äußere Antwortform, die die Struktur und Interaktion des Antwortverhaltens prägt (z. B. Quelltext, Freitext, Auswahl)
- **Repräsentationsformate:** Darstellungsformen innerhalb der Aufgabenstellung, wie etwa Code-Snippets, Diagramme oder Text, die die Art der Informationsaufnahme beeinflussen können
- **Instruktionale Struktur:** ergänzende sprachliche, visuelle oder interaktive Hinweise, die Lernende durch die Aufgabe führen (z. B. Beispiele, Strukturvorgaben, Warnhinweise).

Diese vier Komponenten bilden die Grundlage einer formalisierten Aufgabenbeschreibung, die semantisch interpretierbar und technisch weiterverarbeitbar ist. Durch die semantische Annotation dieser Elemente lassen sich Aufgabenstellungen in ihre didaktisch relevanten Bausteine zerlegen und so beschreiben, dass sie algorithmisch verarbeitet werden können. Der Operator fungiert dabei als semantischer Anker zur kognitiven Dimension, während Aufgabenformat, Repräsentationsformen und Instruktionselemente das Antwortverhalten der Lernenden strukturieren. Erst die Kombination dieser Komponenten mit den inhaltlichen und prozessualen Anforderungen aus Wissens- und Kognitionsdimension ergibt ein vollständiges Aufgabenprofil, das als Grundlage für Diagnose, Adaptivität und Feedbackprozesse dient.

Eine zentrale Anforderung an die Aufgabenformalisierung besteht in der klaren Trennung zwischen der *Oberflächenstruktur* und der *semantischen Struktur* einer Aufgabe. Erstere umfasst die sichtbaren Gestaltungselemente, wie etwa Text, Layout oder begleitende Medien, während Letztere jene Merkmale beschreibt, die die didaktische Intention und den fachlichen Gehalt der Aufgabe bestimmen.

So kann beispielsweise dieselbe Anforderung – „Bestimmen Sie das Minimum eines Feldes“ – in sehr unterschiedlichen sprachlichen Varianten formuliert sein, die sich in Tonalität, Länge oder

Detailgrad unterscheiden, semantisch jedoch identisch sind. Eine formal modellierte Aufgabenstellung abstrahiert daher von ihrer konkreten sprachlichen oder visuellen Gestalt und erfasst stattdessen die didaktisch relevanten Kernmerkmale in strukturierter, maschinenlesbarer Form. Sie bildet somit die Brücke zwischen menschlicher Instruktion und maschineller Interpretation und schafft die Voraussetzung dafür, dass Aufgaben automatisiert analysiert, verglichen und adaptiv weiterverarbeitet werden können.

Für die in Abbildung 5.1 dargestellte Beispielaufgabe zur Bestimmung des Minimums ergibt sich folgende semantische Struktur:

- **Operator:** `implementieren`
  (explizit im Aufgabentext genannt)
- **Aufgabenformat:** `Quellcode`
- **Repräsentationsformate:**
  - `Quellcode`
    (vorgegebener leerer Klassenrumpf `Homework3`)
  - `Text`
    (Einleitender Satz sowie Instruktion)
- **Instruktionale Struktur:**
  - `Instruktion:`
    „Implementieren Sie eine Methode ***`minSearch`****, die das Minimum eines Arrays bestimmt.“*
  - `illustrierendes Beispiel` (optional):
    z. B. Testfälle mit Ein- und Ausgabe (`[3, 5, 1, 4]` → `1`)
  - `Hinweis` (optional):
    z. B. *„Achten Sie auf Sonderfälle wie leere Felder oder negative Werte.“*

Im Fall der `minSearch`-Aufgabe ist der **Operator** `implementieren` explizit in der Aufgabenstellung genannt und legt klar fest, dass die Bearbeitung die produktive Erstellung von Code erfordert. Dieselbe Aufgabenintention kann jedoch auch *implizit* formuliert werden – etwa in der Variante *„Schreiben Sie eine Methode, die das Minimum eines Feldes bestimmt“*. Beide Formulierungen verweisen auf denselben Handlungsauftrag, unterscheiden sich aber in der sprachlichen Oberflächenstruktur.

Das **Aufgabenformat** ist hier als `Quellcode` spezifiziert, da die Aufgabe die Entwicklung einer Methode in Java verlangt. In realen Lernszenarien kann das Format jedoch auch durch den Kurskontext impliziert sein – etwa, wenn eine Serie von Aufgaben in einer Programmierumgebung bearbeitet wird, in der grundsätzlich Quellcode erwartet wird. Auch in solchen Fällen wird das Aufgabenformat semantisch explizit annotiert, um eine eindeutige Zuordnung und Vergleichbarkeit über Aufgaben und Kontexte hinweg zu gewährleisten.

Hinsichtlich der **Repräsentationsformate** kombiniert die Aufgabe textuelle und formale Elemente. Neben der textbasierten Instruktion enthält sie ein Codefragment – den leeren Klassenrumpf `Homework3` – als strukturelles Gerüst für die Lösung. Diese Information könnte alternativ auch rein sprachlich beschrieben werden (z. B. *„Gegeben sei eine leere Klasse* ***`Homework3`****“*), ohne dass Code in die Aufgabenstellung eingebettet wird.

Die hier exemplarisch aufgezeigte **instruktionale Struktur** umfasst neben der eigentlichen Instruktion auch optionale Elemente, die den Bearbeitungsprozess anleiten oder unterstützen können. Ein illustrierendes Beispiel für ein Ein-Ausgabe-Paar wie `[3, 5, 1, 4]` → `1` könnte

etwa als Hilfestellung verfügbar sein, um das erwartete Verhalten der Methode zu verdeutlichen. Dabei zählt nicht nur die reine Existenz eines illustrierenden Beispiels, sondern auch die konkrete Ausgestaltung desselben: Enthält es ein Ein-Ausgabe-Paar mit einem leeren Feld als Eingabe, weist es Lernende implizit auf die Berücksichtigung dieses Sonderfalls hin. Dadurch liefert es einen deutlich stärkeren Impuls für eine robuste Implementierung als ein Beispiel, das ausschließlich positive Zahlen enthält. Alternativ zum Beispiel könnte je nach Kontext auch ein direkter Hinweis in Form einer Instruktion wie *„Achten Sie auf Sonderfälle wie leere Arrays oder negative Zahlen“* erscheinen. Liefert die Methode z. B. bereits für Felder mit Länge > 1 die korrekten Werte, so wäre ein Hinweis auf die Behandlung von Feldern der Länge 0 oder 1 womöglich geeigneter als das zuvor genannte illustrierende Beispiel.

Beide Elemente sind in der ursprünglichen Aufgabenstellung (Abbildung 5.1) zunächst nicht enthalten, gehören jedoch zur semantischen Beschreibung, da sie potenzielle Instruktionselemente darstellen, die als didaktische Erweiterungen fest mit der Aufgabe verknüpft sind. Dadurch kann eine Aufgabe trotz identischer sichtbarer Darstellung didaktisch unterschiedlich ausgestaltet sein, je nachdem, ob solche unterstützenden Elemente vorgesehen sind oder nicht. Die Trennung von *Darstellung* und *Semantik* gewährleistet somit, dass adaptive Systeme den jeweiligen Aufgaben- und Kontextzustand präzise erfassen und darauf aufbauend Feedback dynamisch, konsistent und kontextangemessen generieren können. Dies hat unmittelbare Relevanz für die Generierung kontextsensitiven Feedbacks: Ist ein Hinweis bereits Bestandteil der Aufgabe, kann dieser in adaptiven Systemen für Feedbackentscheidungen einbezogen werden und somit z. B. redundante Rückmeldungen vermieden werden (vgl. `EXFEED` in Kapitel 4).

Technische Anforderungen

Damit Aufgabenstellungen im Rahmen des Y-Modell operationalisiert werden können, sind insbesondere folgende technische Voraussetzungen zu schaffen:

- T1 Ein standardisiertes Aufgabenmodell, das zwischen Darstellung und semantischer Struktur unterscheidet.
- T2 Eine Annotationsebene, in der Operatoren, Konzepte, kognitive Kategorien und Aufgabenformate separat ausweisbar sind.
- T3 Werkzeuge zur konsistenten Annotation bestehender Aufgaben,
- T4 Mechanismen zur Suche, Variation und Sequenzierung von Aufgaben entlang semantischer Merkmale (z. B. alle Aufgaben mit `implementieren` und `array-access`).

Die Aufgabenstellung wird damit zu einem operationalisierten Element, das gleichermaßen menschlich interpretierbar und maschinell verarbeitbar ist. Im Y-Modell übernimmt sie die Funktion, didaktische Intentionen in strukturierter Form sichtbar zu machen und als Steuergröße für diagnostische und adaptive Prozesse zu wirken.

5.3.5 Formalisierung der Antwortdimension

Die *Antwortdimension* beschreibt im Y-Modell die konzeptuelle Verknüpfung zwischen der Aufgabenstellung und dem beobachtbaren Antwortverhalten der Lernenden. Sie fokussiert auf die *reaktiven Ausdrucksformen* der Bearbeitung – also auf Art, Struktur und Qualität der tatsächlich gegebenen Antworten.

Dies geschieht im Y-Modell auf Grundlage des in Abschnitt 5.2 eingeführten Konzepts der *Antwortklasse*, verstanden als kategorial beschreibbare Mengen von Lernendenantworten, die hinsichtlich didaktisch relevanter Kriterien als äquivalent gelten. Damit können nicht nur *richtige*

und *falsche* Antworten unterschieden, sondern auch typische Fehlermuster, alternative Strategien oder stilistische Varianten systematisch erfasst werden.

Antwortklassen ermöglichen es zudem, den *Soll-Zustand* einer Antwort formal zu beschreiben, indem festgelegt wird, welchen Antwortklassen eine korrekte Lösung zugeordnet sein muss bzw. in welchen sie explizit *nicht* enthalten sein darf. Dieser Soll-Zustand ergibt sich unmittelbar aus der Aufgabenbeschreibung: Elemente wie Operator, Aufgabenformat oder Instruktion definieren, welche Arten von Antworten erwartbar sind und welche Merkmale sie aufweisen sollten. So kann etwa überprüft werden, ob eine Aufgabe, die die Implementierung einer Methode fordert, tatsächlich in Form von Quellcode beantwortet wurde, ob geforderte Bestandteile enthalten sind oder ob bestimmte Bearbeitungshinweise aus der Instruktion berücksichtigt wurden.

Durch diese Verknüpfung zwischen Aufgabenbeschreibung und Antwortklassifikation lässt sich der Bewertungs- und Feedbackprozess auf einer abstrakten Ebene operationalisieren, ohne jede individuelle Antwort manuell prüfen zu müssen. Die Analyse erfolgt somit nicht mehr auf der Ebene einzelner Programmtexte, sondern auf der Ebene charakteristischer Antwortmuster. Dies reduziert Komplexität, erhöht Vergleichbarkeit und erleichtert sowohl die algorithmische Verarbeitung als auch die didaktische Interpretation.

Die Formalisierung der Antwortdimension verfolgt damit das Ziel, Lernendenreaktionen nicht nur als bloßes Set von Ergebnissen, sondern als strukturierte Ausdrucksformen kognitiver und fachlicher Prozesse sichtbar und algorithmisch verarbeitbar zu machen. So können differenzierte Analysen ermöglicht und Rückmeldungen an didaktisch bedeutsamen Merkmalen ausgerichtet werden. Die Antwortdimension bildet damit die Brücke zwischen Aufgabenkonstruktion und Lernendenverhalten und macht diese für Diagnose- und Feedbackprozesse nutzbar.

Im Y-Modell werden Antwortklassen als eigenständige Wissenseinheiten aufgefasst, die nicht dem Fachinhalt selbst, sondern der *fachdidaktischen Domäne* eines Fachs zugeordnet sind. Sie kodieren Wissen über typische Bearbeitungsmuster, empirisch nachgewiesene Fehlermuster, Lösungsstrategien oder stilistische Präferenzen innerhalb einer Fachkultur.

Anders als fachliche Konzepte, die in der Wissensdimension verortet sind, beschreiben Antwortklassen keine fachlichen Zielzustände, sondern *epistemische Ausdrucksformen*, wie Lernende auf eine gegebene Aufgabenstellung reagieren können. Dieses fachdidaktische Wissen ist oft erfahrungsbasiert bzw. empirisch theoretisch hergeleitet und kann zur Gestaltung von Feedback-Strategien, Diagnosemodellen und adaptiven Lernpfaden genutzt werden. Die Repräsentation dieser Klassen erfolgt als strukturierte Einheiten mit zugeordneten Metadaten und semantischer Verknüpfung zur jeweiligen Aufgabe.

Technische Anforderungen

Damit die Antwortdimension operabel wird, sind folgende Voraussetzungen zu schaffen:

- A1 Mechanismen zur semantischen Annotation von Aufgaben mit Antwortklassen
- A2 Verfahren zur automatisierten Analyse und Zuordnung von Antworten zu Antwortklassen
- A3 Funktionen zur Definition des **Soll-Zustands** einer Aufgabenlösung durch die Kombination von Antwortklassen, einschließlich der Kennzeichnung, welche Klassen *vorhanden sein müssen* bzw. *nicht auftreten dürfen*

Die Antwortdimension schließt damit die Lücke zwischen Aufgabenbeschreibung und beobachtetem Lernendenverhalten. Im Zusammenspiel mit Wissens-, kognitiver und Aufgabenstellungsdimension ermöglicht sie es, Antworten nicht nur zu bewerten, sondern sie systematisch als Indikatoren für Diagnose und adaptive Unterstützung nutzbar zu machen.

5.4 Diskussion

Die Modellierung der Aufgaben entlang der Dimensionen Wissen, Kognition, Aufgabenstellung und Antwort schafft eine einheitliche Struktur, um Lernprozesse formal zu beschreiben. Besonders das Zusammenspiel dieser Dimensionen im Y-Modell bietet einen klaren Rahmen, in dem Aufgaben- und Antwortelemente systematisch miteinander verknüpft werden können. Die empirischen Ergebnisse aus der Fallstudie (vgl. Abschnitt 5.2.5) zeigen, dass sich durch die Kombination dieser Komponenten ein hohes Maß an Standardisierung, Transparenz und Konsistenz erzielen lässt. Lernendenantworten können dadurch differenzierter analysiert und Feedbackprozesse gezielter gesteuert werden.

Dem stehen jedoch gewisse Grenzen gegenüber. Die erstmalige Annotation von Aufgaben mit den erforderlichen Metadaten und Antwortklassen ist mit einem hohen initialen Aufwand verbunden. Dieser Aufwand betrifft insbesondere die Operationalisierung der Wissens- und Aufgabendimension sowie die präzise Definition des erwarteten Soll-Zustands. Gleichzeitig deutet sich ein deutlicher Nutzen an: Einmal erstellte Modellinstanzen können wiederverwendet, erweitert und in unterschiedlichen Kontexten adaptiert werden, was insbesondere bei standardisierten oder wiederkehrenden Aufgabenformaten Effizienzgewinne verspricht (vgl. Abschnitt 5.2.4.2). Zudem ermöglicht die interoperable Struktur des Modells die Integration in bestehende Assessment- und Feedbacksysteme, etwa durch die Nutzung gemeinsamer Datenformate oder Ontologien. Die Modellierung kann somit als strategische Investition verstanden werden, die langfristig Skalierbarkeit und Konsistenz im Feedbackprozess erhöht sowie vergleichbare Daten für empirische Analysen generiert.

Die für Antwortklassen formulierten Anforderungen der Objektivität und Interpretationsfreiheit (vgl. Abschnitt 5.2) bilden die theoretische Grundlage für deren zuverlässige und algorithmische Nutzbarkeit. Die Bedingung der Objektivität ist für manche Antwortklassen, insbesondere solche die stilistische Merkmale wie Lesbarkeit, Strukturierung oder Namenskonventionen betrifft, nur eingeschränkt umsetzbar. Ob eine Variable „sprechend" benannt ist oder Quellcode gut strukturiert erscheint, entzieht sich einer vollständig objektiven Beurteilung, da solche Einschätzungen immer auch soziale und kontextuelle Normen reflektieren.

Die Forderung nach Interpretationsfreiheit – also der Verzicht auf Zuschreibungen mentaler Zustände – grenzt das Modell bewusst ein. Diese Beschränkung dient der Schaffung einer transparenten und technisch anschlussfähigen Beschreibungsebene, auf der Lernendenantworten zunächst rein beobachtungsbasiert klassifiziert werden. Damit wird ein interpretierbarer, datengetriebener Ausgangspunkt geschaffen, der in einem zweiten Schritt durch lernendenzentrierte Modelle oder empirische Analysen ergänzt werden kann. So bleibt das System offen für zukünftige Forschung, die sich etwa mit der Erkennung von Fehlvorstellungen oder motivationalen Zuständen befasst, ohne die methodische Strenge der aktuellen Modellschicht zu untergraben. Das Modell ist somit nicht als vollständige Beschreibung kognitiver Prozesse, sondern als strukturierter Rahmen zur Beobachtung und Diagnose des Antwortverhaltens zu verstehen.

Das im Kapitel vorgestellte Y-Modell bildet die konzeptionelle Rahmung für die formale Beschreibung von Aufgaben und Antworten. Seine Stärke liegt in der klaren Trennung und gleichzeitigen Verknüpfung der Wissens-, kognitiven, Aufgaben- und Antwortdimension. Diese Struktur erlaubt eine systematische Abbildung und Erforschung von Lernprozessen, insbesondere im Bereich der Programmierung, wo Aufgaben in der Regel sowohl kognitive als auch prozedurale Transformationen erfordern. Die formale Definition der Antwortklassen ermöglicht hierbei, Bearbeitungsverläufe nicht nur qualitativ zu interpretieren, sondern auch quantitativ und algorithmisch zu erfassen.

Gleichwohl ist das Y-Modell nicht ausschließlich auf den Bereich der Programmierung beschränkt. Seine abstrakte Grundstruktur – die Transformation eines gegebenen Ausgangs- in einen Zielzustand unter Berücksichtigung von Wissens- und Aufgabenkontext – ist auf viele weitere domänenspezifische Lernprozesse übertragbar. Besonders in Bereichen mit klar definierbaren Lösungsräumen, etwa in Mathematik, Logik oder den Naturwissenschaften, bietet das Modell einen geeigneten Rahmen zur systematischen Beschreibung und Analyse von Aufgabenbearbeitungen. Grenzen ergeben sich dort, wo Lernprodukte stark kreativ, diskursiv oder sozial eingebettet sind, etwa in literarischen oder gestalterischen Fächern. In solchen Kontexten ist eine formale Operationalisierung der Antwortdimension möglicherweise nur eingeschränkt realisierbar.

Zusammenfassend lässt sich festhalten, dass das in diesem Kapitel entwickelte Modell einen tragfähigen theoretischen und technischen Rahmen für die Formalisierung von Aufgaben- und Antwortstrukturen bereitstellt. Die Prinzipien der Objektivität und Interpretationsfreiheit gewährleisten methodische Strenge und algorithmische Anschlussfähigkeit, während das Y-Modell als integrierende Struktur die Interoperabilität zwischen verschiedenen didaktischen und technologischen Komponenten sichert. Die Grenzen liegen primär im initialen Aufwand der Modellierung und in der eingeschränkten Objektivierbarkeit bestimmter Antwortdimensionen. Dennoch zeigt sich insgesamt, dass die Formalisierung nicht nur eine theoretische Struktur liefert, sondern konkrete, empirisch überprüfbare Potenziale für die Automatisierung und Qualitätssicherung von Feedbackprozessen in Programmierlernsystemen eröffnet.

Kapitel 6

Potenziale großer Sprachmodelle für kontextsensitive Feedbackprozesse

Wie in Kapitel 4 gezeigt, berücksichtigen Lehrpersonen in ihren Entscheidungen eine Vielzahl individueller und situativer Faktoren und nutzen zudem ein breites Spektrum an Feedback-Typen. Diese Vielfalt automatisiert in Lernsystemen abzubilden, stellt bislang jedoch eine große Herausforderung dar: Selbst bei wohldefinierten Aufgaben ergibt sich durch die Kombination dieser Faktoren eine nahezu unüberschaubare Vielzahl relevanter Kontexte. Für diese Vielfalt ist eine vollständig vordefinierte Rückmeldestruktur – etwa in Form von Templates oder regelbasierten Textbausteinen – praktisch nicht realisierbar. Dies dürfte einer der Gründe sein, weshalb viele bestehende Systeme vorwiegend fehlerorientiertes, einfaches Feedback bereitstellen (Keuning, Jeuring und Heeren 2019; Jeuring et al. 2022).

Mit der zunehmenden Verfügbarkeit leistungsfähiger generativer KI-Systeme eröffnen sich hier neue Potenziale, Rückmeldungen kontextsensitiv, variantenreich und typenspezifisch steuerbar zu generieren. Große Sprachmodelle sind primär für die Kommunikation mit Menschen konzipiert und zeigen bemerkenswerte Leistungen in der Verarbeitung natürlicher Sprache, der Generierung kohärenter Texte sowie in dialogischen Interaktionen (Chang et al. 2024; Chiang et al. 2024). Erste Studien zur Anwendung in der Programmierausbildung weisen bereits auf Potenziale für die Generierung von Code-Erklärungen und Fehlermeldungen hin (Azaiz, Kiesler und Strickroth 2024; Leinonen et al. 2023b; Sarsa et al. 2022; MacNeil et al. 2023; Bengtsson und Kaliff 2023), bleiben jedoch in der Regel auf einfache Hilfeanfragen beschränkt und berücksichtigen die Vielfalt möglicher Feedbacktypen bislang kaum.

Das Fachgebiet der Programmierung bietet für den Einsatz solcher Modelle in Bezug auf Feedback günstige Voraussetzungen: Typische Fehler, wiederkehrende Lösungsmuster und domänenspezifisches Wissen sind in großer Zahl in frei zugänglichen Quellen dokumentiert – etwa in Foren, Tutorials, Lehrvideos, Übungsplattformen oder Open-Source-Repositories. Diese breite Verfügbarkeit geht auf eine spezifische Fachkultur zurück, in der es üblich ist, Probleme und Lösungen öffentlich zu diskutieren, beispielsweise in Foren wie StackOverflow. Auch wenn die genauen Trainingsdaten vieler aktueller Sprachmodelle nicht veröffentlicht wurden, ist plausibel, dass solche Materialien in erheblichem Umfang Eingang in die Modelle gefunden haben. Damit liegt die Annahme nahe, dass typische Aufgaben- und Fehlermuster in den Parametern von LLMs statistisch repräsentiert sind und sich bei geeigneter Promptgestaltung für die Erzeugung plausibler Rückmeldungen nutzen lassen.

Teile dieses Kapitels wurden bereits vorab veröffentlicht in Kiesler, Lohr und Keuning (2023) sowie Lohr, Keuning und Kiesler (2025). Für Details siehe Übersicht zu Vorabveröffentlichungen auf Seite VII.

Offen bleibt jedoch, unter welchen Bedingungen LLMs qualitativ hochwertiges, didaktisch anschlussfähiges Feedback generieren können – insbesondere solches, das sich an etablierten Typologien orientiert und systematisch auf unterschiedliche Antwortvarianten reagiert. Ebenso ungeklärt ist, inwiefern sich LLMs gezielt zur Erzeugung spezifischer Feedbacktypen steuern lassen – ohne inhaltlich irreführende Informationen oder ungewollte Hilfestellungen zu erzeugen.

Vor diesem Hintergrund wird in diesem Kapitel untersucht, inwieweit sich große Sprachmodelle für die Generierung kontextsensitiven Feedbacks in der Programmierausbildung eignen (**FF3**, vgl. Abschnitt 3.2). Aufbauend auf dem in Kapitel 5 entwickelten formalen Bezugsrahmen wird geprüft, wie sich durch explizite Kontextbereitstellung differenzierte und konsistente Rückmeldungen ableiten lassen. Im Zentrum steht dabei die Frage nach der Qualität der generierten Rückmeldungen sowie nach ihrer Ausrichtung an unterschiedlichen Feedback-Typen. Methodisch ist dieses Vorhaben im Rahmen des Design Science Research-Prozesses (vgl. Abschnitt 3.3.1) im *Design Cycle* verortet: Ziel ist es, ein prototypisches Artefakt zu entwickeln und zu erproben, welches die automatische Generierung typenspezifischer Feedbacknachrichten ermöglicht, die grundlegenden Qualitätsansprüchen genügen.

6.1 Stand der Forschung

Seit der öffentlichen Verfügbarkeit von ChatGPT im November 2022 hat sich in Wissenschaft, Lehre, Medien und Gesellschaft eine intensive Debatte über die Potenziale, Risiken und Auswirkungen großer Sprachmodelle entfacht.

Parallel dazu haben LLMs in nahezu allen Forschungsdisziplinen stark an Aufmerksamkeit gewonnen (Fan et al. 2024). In der Informatik – insbesondere in den Bereichen der Künstlichen Intelligenz und der Computerlinguistik – waren Sprachmodelle bereits lange vor der Veröffentlichung von ChatGPT ein etabliertes Forschungsfeld. Ihre Entwicklung und Evaluation erfolgte dabei überwiegend innerhalb dieser Disziplinen und blieb über viele Jahre ein technisches Thema, das vor allem Fragen der Modellarchitektur, Trainingsverfahren und Leistungsbewertung adressierte. Erst mit dem breiten öffentlichen Zugang zu generativen Systemen Ende 2022 rückten ihre Anwendungen und Implikationen auch in anderen Wissenschaftsbereichen und gesellschaftlichen Kontexten in den Fokus. Seit einigen Jahren wächst die Forschung zu Einsatzmöglichkeiten, ethischen Implikationen und Evaluationsmethoden nun auch in Geistes-, Sozial- und Bildungswissenschaften, einschließlich Arbeiten im Kontext der Programmierausbildung (Prather et al. 2023a; Prather et al. 2025; Becker et al. 2024).

Bibliometrische Analysen verdeutlichen die rasante Expansion der Publikationstätigkeit zu Chatbots und LLM-bezogenen Themen (Fan et al. 2024), mit jährlichen Wachstumsraten von bis zu 27 % in den Datenbanken *Web of Science* und *Scopus* (Khosravi et al. 2024). Auch aktuelle Übersichtsarbeiten (Prather et al. 2023b; Prather et al. 2025; Becker et al. 2024) verweisen auf die hohe Dynamik des Feldes und betonen den Bedarf an empirisch fundierten, kontextsensitiven Untersuchungen. Vor diesem Hintergrund gestaltet es sich als Herausforderung, den Forschungsstand umfassend zu erfassen – insbesondere, da im Bereich der Programmierausbildung bislang wenige systematische Literaturübersichten (SLRs) vorliegen, die LLM-Anwendungen im Kontext des Programmierlernens oder der Informatikdidaktik konsolidieren. Der folgende Überblick erhebt daher keinen Anspruch auf Vollständigkeit, sondern zielt darauf, zentrale Entwicklungen und Forschungsrichtungen zur Nutzung großer Sprachmodelle in der *Programmierausbildung* und insbesondere im Bereich der *automatisierten Feedbackbereitstellung* zu skizzieren.

Becker et al. (2023) diskutieren zentrale Chancen und Herausforderungen, die sich aus der Integration großer Sprachmodelle in Bildungskontexte ergeben. Sie betonen, dass die rasante Entwicklung generativer Systeme dazu zwingt, bisherige Bildungspraktiken im Lichte dieser neuen

Technologien zu überprüfen. Diese Forderung markiert einen grundlegenden Paradigmenwechsel: Während LLMs zunächst als technologische Innovation betrachtet wurden, rückt zunehmend ihre Funktion für Lehre und Lernen in den Mittelpunkt – etwa als Unterstützungs-, Diagnose- oder Feedbackinstrument.

In der Programmierausbildung eröffnet sich damit eine Vielzahl potenzieller Einsatzszenarien. Diskutiert werden unter anderem die Unterstützung bei der Codegenerierung, die Erklärung und Kommentierung von Programmlogik, die Identifikation und Behebung von Fehlern, die automatisierte Bewertung von Programmieraufgaben sowie tutorielle oder dialogbasierte Lernbegleitung. Diese Szenarien spiegeln unterschiedliche didaktische Zielrichtungen wider – von der Reduktion kognitiver Belastung bei Routineaufgaben bis hin zur Förderung metakognitiver Reflexion durch interaktive Erklärungen.

Frühe Forschungsarbeiten konzentrierten sich zunächst stark auf die Frage, inwieweit Sprachmodelle überhaupt in der Lage sind, Programmieraufgaben korrekt zu lösen. Einer der ersten Beiträge in diesem Bereich erschien Anfang 2022: Finnie-Ansley et al. (2022) analysierten die Leistungsfähigkeit des OpenAI-Codex-Modells bei typischen CS1-Programmierübungen und zeigten, dass es in vielen Fällen mit den Leistungen realer Studierender vergleichbar war. Eine anschließende Folgestudie zu CS2-Aufgaben bestätigte diese Befunde und dokumentierte erneut beeindruckende Resultate (Finnie-Ansley et al. 2023). Diese frühen Arbeiten legten den Schwerpunkt auf Output-Qualität und Problemlösekompetenz des Modells, weniger jedoch auf dessen didaktischen Nutzen oder Auswirkungen auf Lernprozesse.

Darauf aufbauend rückten in neueren Studien zunehmend Aspekte der Interaktion und Steuerung in den Vordergrund. Denny, Kumar und Giacaman (2023) untersuchten etwa, wie Studierende Eingabeaufforderungen (Prompts) beim Einsatz von GitHub Copilot formulieren und weiterentwickeln. Ziel war es, Aufgabentypen zu identifizieren, bei denen Copilot an seine Grenzen stößt, und zu verstehen, wie präzise und kontextualisierte Prompts gestaltet sein müssen, um qualitativ hochwertige Codevorschläge zu erhalten. Damit verschob sich die Perspektive von der reinen Leistungsbewertung hin zur Erforschung von Strategien, mit denen Lernende die Leistungsfähigkeit von LLMs effektiv nutzen können.

Parallel dazu erschienen zahlreiche Studien, die die Leistungsfähigkeit von LLMs im Kontext der Beantwortung von Prüfungsfragen und Assessments in Programmierkursen untersuchten (Savelka et al. 2023; Joshi et al. 2024). Diese Arbeiten liefern wertvolle empirische Hinweise darauf, in welchem Maße Sprachmodelle domänenspezifisches Wissen reproduzieren und strukturierte Aufgabenformate bearbeiten können. Zugleich verdeutlichen sie, dass eine rein output-orientierte Bewertung die pädagogischen Potenziale generativer Modelle nur unzureichend erfasst – insbesondere, wenn es um ihre Rolle bei der Unterstützung individueller Lernprozesse und der kontextsensitiven Feedbackbereitstellung geht.

Nachdem frühe Arbeiten insbesondere darauf abzielten, die Leistungsfähigkeit großer Sprachmodelle bei der *eigenständigen Lösung von Programmieraufgaben* zu untersuchen (Finnie-Ansley et al. 2022; Denny, Kumar und Giacaman 2023; Wermelinger 2023; Cipriano und Alves 2023; Kiesler und Schiffner 2023a), verlagert sich der Forschungsschwerpunkt zunehmend auf ihren Einsatz zur *Unterstützung des Lernens* und zur *didaktischen Integration in den Unterricht* (Scholl, Schiffner und Kiesler 2024; Prather et al. 2024). Während in der ersten Forschungsphase die Frage im Vordergrund stand, ob LLMs Programmieraufgaben korrekt lösen können, rückt nunmehr die Untersuchung ihrer Rolle als *Assistenzsysteme* in Lernprozessen in den Fokus – insbesondere bei der Fehlersuche, Code-Reparatur und Erklärung von Programmverhalten.

Zhang et al. (2024) wendeten das Codex-Completion-Modell auf Python-Programme von Studierenden an und konnten zeigen, dass ein auf Code trainiertes Sprachmodell sowohl syntaktische als auch semantische Fehler beheben kann. Aufbauend darauf kombinierten Ishizue et al. (2024)

das etablierte Tool *Refactory* mit den GPT-Modellen von OpenAI, um fehlerhafte Studierendenlösungen effizienter zu korrigieren. Die Integration des LLMs ermöglichte die Bearbeitung von Programmen mit Syntaxfehlern und unvollständigen Vergleichsdaten und erhöhte damit signifikant die Erfolgsrate automatischer Reparaturprozesse.

Kazemitabaar et al. (2023) gingen darüber hinaus der Frage nach, wie sich der Zugang zu solchen Werkzeugen auf das Lernverhalten auswirkt. In einer kontrollierten Studie erhielten die Hälfte Programmieranfängerinnen und -anfänger während der Aufgabenbearbeitung Zugang zu Codex. Die Ergebnisse zeigen, dass die Nutzung des Modells die Codequalität und Bearbeitungseffizienz im Vergleich zu Lernenden, die keinen Zugang hatten, deutlich steigerte, zugleich aber die Art der Aufgabenbearbeitung veränderte: Lernende interagierten weniger explorativ und stärker zielorientiert, was auf neue Formen der Abhängigkeit von automatischer Unterstützung hinweist.

Ein weiterer Forschungsstrang befasst sich mit der Verbesserung von Compiler-Fehlermeldungen und Code-Erklärungen – zentrale Herausforderungen insbesondere für Programmieranfängerinnen und -anfänger. Unverständliche Fehlermeldungen gelten als häufige Quelle von Frustration und Abbruchmotivation; hier bieten LLMs neue Möglichkeiten zur sprachlich und didaktisch optimierten Rückmeldung.

Wang, Mitchell und Piech (2024) führten eine groß angelegte randomisierte Kontrollstudie durch, in der sechs verschiedene Varianten von Fehlermeldungen verglichen wurden, darunter zwei GPT-3-basierte Ansätze. Die Ergebnisse zeigen, dass die von GPT generierten Fehlermeldungen am effektivsten waren: Studierende behoben Fehler schneller und wiederholten sie seltener. Ähnliche Ergebnisse berichten Taylor et al. (2024), die ein in einen `C`-Compiler integriertes LLM-basiertes System entwickelten, das studierendenfreundliche Fehlermeldungen generiert und großflächig erprobten. Beide Studien verdeutlichen, dass LLMs nicht nur technische Diagnosen liefern, sondern auch die kommunikative Qualität von Fehlermeldungen verbessern können.

Kimmel et al. (2024) untersuchten die Wirkung von `GPT-4`-basiertem Feedback zu Compiler-, Laufzeit- und Logikfehlern in einem automatisierten Bewertungssystem. Ihre Ergebnisse zeigen, dass LLM-gestützte Rückmeldungen zu effizienteren Korrekturprozessen führen können, die Wahrnehmung des Feedbacks jedoch stark vom *Interface-Design* und der *Erwartungshaltung der Lernenden* abhängt – Faktoren, die für die Gestaltung effektiver Feedbacksysteme von zentraler Bedeutung sind.

Neben der Fehlerbehebung werden LLMs zunehmend auch zur Erklärung von Programmcode eingesetzt. MacNeil et al. (2022b) und Sarsa et al. (2022) demonstrierten, dass generierte Code-Erklärungen qualitativ mit menschlichen Erklärungen vergleichbar sind und insbesondere für Anfängerinnen und Anfänger eine wertvolle Orientierung bieten. MacNeil et al. (2023) zeigten, dass Studierende zeilenweise generierte Code-Erklärungen als hilfreich und lernförderlich empfinden. Eine Folgestudie von Leinonen et al. (2023a) bestätigte diese Befunde: Die Erklärungen des LLM wurden hinsichtlich Verständlichkeit und Genauigkeit besser bewertet als von Studierenden erstellte Erklärungen, ohne dass eine ablehnende Haltung gegenüber dem nicht-menschlichen Ursprung des Feedbacks erkennbar war.

Insgesamt zeichnet sich ein Übergang von der reinen *Lösungsproduktion* hin zur *Unterstützung bei Diagnose, Reparatur und Erklärung* ab. Damit wird der Anschluss an die *Feedbackgenerierung* vorbereitet: Statt die Generierung korrekter Endprodukte zu untersuchen, wurde zunehmend erforscht, inwiefern Systeme sich für Zwischendiagnosen, Hinweise und erklärungsbasierte Anleitungen eignen – zentrale Bausteine formativer Unterstützung.

In den letzten Jahrzehnten wurden vielfältige Verfahren zur automatisierten Feedbackerzeugung entwickelt, um Lehrende und Lernende in großem Umfang zu unterstützen – insbesondere in skalierenden Settings wie MOOCs (Messer et al. 2024; Paiva, Leal und Figueira 2022; Keuning,

Jeuring und Heeren 2019). Neben regelbasierten und testgetriebenen Ansätzen wurde jüngerer Zeit auch untersucht, inwiefern große Sprachmodelle sich zur Generierung *variantenreicher* und *sprachlich adaptiver* Rückmeldungen eignen:

Erste Studien charakterisieren die Qualität LLM-generierten Feedbacks in authentischen Aufgaben. Balse et al. (2023) nutzten Abschlussabgaben zu sieben Übungen als Eingabe für `GPT-3` und bewerteten Korrektheit und Feedbackart: 71 % der Rückmeldungen wurden korrekt als richtig klassifiziert. Etwa 62 % enthielten zutreffende Kritik bzw. Vorschläge. Identifiziert wurden vier dominante Themen: „allgemeines Feedback", „Variableninitialisierung/-aktualisierung", „Bedingungen" und „Iteration".

Pankiewicz und Baker (2023) integrierten `GPT-3.5` in ein bestehendes Bewertungswerkzeug für `C#`-Einreichungen (38 Übungen). Studierende beurteilten das Feedback mittels Likert-Skalen. Zusätzlich wurden Leistung, Bearbeitungszeit und affektive Variablen erfasst und mit einer Kontrollgruppe ohne Hinweise verglichen. Nahezu die Hälfte bewertete das Feedback positiv – einige der Studierenden waren jedoch mit den Hinweisen weniger zufrieden. Die Gruppe mit `GPT`-Hinweisen löste Aufgaben schneller und reichte häufiger korrekte Lösungen ein. In nachfolgenden Aufgaben ohne Hinweise taten sich diese Studierenden zunächst schwer, holten jedoch später zur Kontrollgruppe auf. Der Prompt enthielt die Übungsbeschreibung, den Code des Studierenden, Informationen über fehlgeschlagene Tests und Compilerfehler sowie basierend auf letzteren einen Prompt mit Anweisungen. In der Arbeit wurde ein Beispiel für diesen Prompt gegeben, in dem darauf hingewiesen wurde, die Lösung nicht preiszugeben und nur Hinweise zu geben sowie bestimmte Elemente (z. B. Schlüsselwörter, Variablennamen, Zeilennummern) mit HTML-Tags hervorzuheben.

Einen zweiten Strang bilden Arbeiten zu Prompt-Strategien und Qualitätskriterien: Roest, Keuning und Jeuring (2024) entwickelten iterativ Prompts zur Generierung *nächster Schritte* für einführende `Python`-Übungen (`GPT-3.5`) und setzten diese in einem Online-Tool ein. Erfahrene Lehrpersonen bewerteten die Hinweise entlang validierter Kriterien (Art, Detailgrad, Information, Personalisierung, Angemessenheit, Spezifität, irreführende Information, Ton, Länge). Insgesamt fiel die Bewertung der Rückmeldungen positiv aus. Defizite zeigten sich u. a. bei Detailtiefe und gelegentlichen Fehlinformationen, insbesondere nahe der Zielerreichung. Xiao, Hou und Stamper (2024) untersuchten die Wirkung mehrerer Hinweisstufen – von allgemeinen Textimpulsen bis zu konkreten Codevorschlägen.

Mit Blick auf Modellgenerationen zeigen Vergleichsstudien, dass sich die Feedbackqualität verbessert, zugleich aber neue Herausforderungen entstehen: Azaiz, Deckarm und Strickroth (2023) evaluierten Rückmeldungen von `text-davinci-003` für zwei `Java`-Übungen und teilten Code-Einreichungen in drei Kategorien ein: (1) syntaktisch und funktional korrekt (SCFC), (2) syntaktisch korrekt, aber funktional nicht korrekt (SCFI, entspricht nicht den Aufgabenanforderungen) und (3) syntaktisch und funktional inkorrekt (SIFI). Berichtet werden u. a. Anforderungslücken und inadäquate Vorschläge – mit dem Fazit, dass LLM-Rückmeldungen (noch) nicht für summative Benotung geeignet sei, wohl aber potenziell als Assistenz für Tutorinnen und Tutoren. In einer Folgestudie mit demselben Datensatz (55 Studierendenlösungen) zeigten Azaiz, Kiesler und Strickroth (2024) für `GPT-4 Turbo` deutliche Verbesserungen hinsichtlich Personalisierung und Genauigkeit. Die vollständige Korrektheit aller Feedbackdetails lag jedoch bei 52 %. Dabei ist hervorzuheben, dass das Befolgen sämtlicher Vorschläge in fast allen Fällen (bis auf zwei) zu korrekten Lösungen geführt hätte.

Nguyen und Allan (2024) präsentieren ihre Erfahrungen mit der Verwendung von `GPT-4`, um abgestuftes, formatives Feedback zu `Java`-Code und Pseudocode im Rahmen eines Kurses über Algorithmen und Datenstrukturen zu geben. Ziel der Untersuchung war es, Feedback zum konzeptionellen Verständnis, zur Syntax und zur Zeitkomplexität sowie Hinweise oder Leitfragen für

den nächsten Schritt zu erhalten. Sie analysierten das Feedback hinsichtlich seiner Genauigkeit und Nützlichkeit sowie seiner Korrektheit und kamen zu dem Schluss, dass `GPT-4` im Allgemeinen genaues Feedback lieferte, das als nützlich für die nächsten Schritte der Studierenden empfunden wurde. Sie stellten jedoch auch eine unterschiedliche Leistung bei der Ausgabe der 113 Einreichungen zu vier Programmierproblemen fest.

Methodisch weiterführend kombinieren Phung et al. (2024) symbolische Artefakte (fehlgeschlagene Tests, Referenzprogramm) mit LLM-Generierung und einem Validierungsschritt zur Qualitätssicherung, um Feedback zu fehlerhaften Programmen zu erzeugen – ein Beispiel für hybride Pipelines. Schließlich untersuchten Koutcheme et al. (2024) Open-Source-LLMs zur Feedbackgenerierung und nutzten wiederum `GPT` zur automatischen Qualitätsbewertung der erzeugten Rückmeldungen.

Die dargestellten Untersuchungen verdeutlichen, dass große Sprachmodelle in der Programmierausbildung bereits in vielfältiger Weise erprobt werden – von der Codegenerierung über Fehlersuche und Erklärungen bis hin zu ersten Ansätzen der automatisierten Feedbackerzeugung. Gleichwohl bleibt weitgehend ungeklärt, *wie* solche Rückmeldungen didaktisch fundiert, kontextsensitiv und zugleich systematisch steuerbar bereitgestellt werden können.

Besonders die Frage der *Steuerbarkeit* generativer Systeme im Hinblick auf unterschiedliche Feedbacktypen ist bislang nur ansatzweise untersucht. Zwar existieren erste Studien, die verschiedene Prompt-Strategien zur Beeinflussung des Modelloutputs erproben (Roest, Keuning und Jeuring 2024; Xiao, Hou und Stamper 2024), doch bleibt offen, in welchem Maße sich die erzeugten Rückmeldungen gezielt an bestehenden didaktischen Typologien orientieren oder adaptiv auf verschiedene Antwortvarianten reagieren lassen. Ebenso ungeklärt ist, wie sich die Qualität und Kohärenz der generierten Rückmeldungen verändert, wenn Aufgaben- und Antwortkontext explizit einbezogen werden – insbesondere im Hinblick auf die Reduktion irreführender oder sachlich fehlerhafter Inhalte.

Im Rahmen des folgenden Kapitels werden diese offenen Fragen empirisch aufgegriffen. Hierzu wird aufbauend auf den in Kapitel 4 identifizierten Merkmalen menschlicher Rückmeldungen und der in Kapitel 5 entwickelten formalen Beschreibung von Aufgaben- und Antwortstrukturen wird untersucht, inwiefern große Sprachmodelle zur automatisierten Generierung von Feedback eingesetzt werden können.

Dazu werden zwei komplementäre Studien vorgestellt, die den Einfluss des verfügbaren Kontextes auf die Qualität und Passung generierter Rückmeldungen analysieren. Die erste Studie untersucht, *wie* die von LLMs erzeugten Rückmeldungen ausgestaltet sind, wenn dem Modell lediglich die Lernendenantwort ohne zusätzlichen Aufgaben- oder Kontextbezug zur Verfügung steht. Die zweite Studie erweitert diesen Ansatz um den im Y-Modell beschriebenen Aufgaben- und Antwortkontext, um zu prüfen, *in welchem Maße* sich dadurch Qualität, Spezifität und didaktische Angemessenheit der Rückmeldungen verbessern.

6.2 Generierung von Rückmeldungen ohne kontextuelle Einbettung

Die erste empirische Untersuchung verfolgt das Ziel, die Charakteristika von Feedbacknachrichten zu analysieren, die ein großes Sprachmodell generiert, wenn Lernende lediglich eine Programmierlösung ohne zusätzliche Kontextinformationen übermitteln. Ausgangspunkt ist die Annahme, dass in realistischen Nutzungsszenarien – etwa bei spontanen Interaktionen mit Tools wie ChatGPT – häufig nur ein minimaler Prompt gegeben wird, der weder die ursprüngliche Aufgabenstellung noch die intendierten Lernziele explizit enthält. Dabei wird bewusst davon

ausgegangen, dass keinerlei Prompting-Strategien (z. B. explizite Rolleninstruktionen, strukturierte Fragen oder schrittweise Eingaben) zum Einsatz kommen.

In diesem Szenario wird untersucht, wie ein LLM auf Anfragen vom Typ *„What's wrong with my code?"* reagiert, wenn als einziger Kontext die studentische Lösung (der Programm-Code) bereitgestellt wird. Im Zentrum stehen dabei folgende Aspekte: (a) die inhaltlichen Merkmale und die formale Qualität der Rückmeldungen, (b) das Auftreten potenzieller metakognitiver oder motivationaler Elemente, sowie (c) typische Fehlermuster oder irreführende Informationen. Darüber hinaus interessiert, inwiefern bestimmte Rückmeldetypen spontan auftreten und wie stark die generierten Antworten zwischen verschiedenen Modellläufen variieren. Auf Grundlage dieser Zielsetzung wird die in Kapitel 3 formulierte Forschungsfrage **FF3.1** aufgegriffen, die adressiert, welche Charakteristika mittels großer Sprachmodelle generierte Rückmeldungen aufweisen und wie sich deren Qualität in Abhängigkeit vom bereitgestellten Kontext verändert. Die vorgestellte Untersuchung bildet dabei den ersten Teil der Beantwortung dieser Frage, indem sie Rückmeldungen in einem kontextarmen Setting analysiert. In der zweiten Untersuchung (siehe Abschnitt 6.3) wird das Setting systematisch erweitert, um Unterschiede in der Qualität der Rückmeldungen in Abhängigkeit vom bereitgestellten Kontext sichtbar zu machen.

Die Untersuchung ist explorativ angelegt und dient als erster Schritt zur systematischen Charakterisierung von LLM-generierten Rückmeldungen in solchen realitätsnahen Interaktionssituationen. Damit soll einerseits das Potenzial großer Sprachmodelle sichtbar werden, auch ohne explizite Kontextinformationen elaborierte und vielfältige Feedbackelemente zu erzeugen. Andererseits werden typische Grenzen, Inkonsistenzen und irreführende Informationen herausgearbeitet, die in diesen Nutzungsszenarien auftreten können. Auf diese Weise liefert die Studie die Grundlage für die anschließende Untersuchung, in der ein erweiterter Kontext berücksichtigt wird, sodass die beiden Studien gemeinsam eine differenzierte Beantwortung von **FF3.1** ermöglichen.

6.2.1 Methodisches Vorgehen

Zur Beantwortung der Forschungsfrage **FF3.1** wurde ein qualitatives Studiendesign gewählt, das sich am Verfahren der deduktiven Kategorienanwendung im Rahmen der strukturierenden Inhaltsanalyse nach Mayring (2015a) orientiert (siehe Abschnitt 3.3.2). Im Mittelpunkt steht die Analyse von Rückmeldungen, die ein großes Sprachmodell (ChatGPT, Version März 2023) zu studentischen Programmierlösungen generiert, wenn lediglich ein minimaler Prompt in Form einer allgemeinen Hilfsanfrage bereitgestellt wird.

Das methodische Vorgehen umfasst zwei zentrale Phasen:

1. **Erhebung der LLM-generierten Rückmeldungen:** Als Ausgangsmaterial werden Programmierlösungen aus einem Datensatz realer Studierendensubmissions ausgewählt, die typische Fehlermuster des Anfangsunterrichts enthalten. Diese Lösungen wurden dem Sprachmodell ohne zusätzliche Kontextinformationen übergeben. Für jede Lösung wurden mehrere Rückmeldungen generiert. Sowohl die Auswahlkriterien der Studierendenlösungen als auch die Interaktionsparameter mit dem Modell wurden zuvor systematisch festgelegt (siehe Abschnitt 6.2.1.1).

2. **Qualitative Inhaltsanalyse:** Anschließend werden die generierten Rückmeldungen inhaltsanalytisch ausgewertet. Grundlage hierfür bildet ein deduktiv entwickeltes Kategoriensystem, das sich aus bestehenden theoretischen Feedbacktypologien speist und zentrale inhaltliche, qualitative sowie metakognitive Merkmale erfasst (Narciss 2007; Keuning, Jeuring und Heeren 2019). Die Kodierung erfolgte durch zwei unabhängige Personen, wobei Uneinigkeiten im Konsensverfahren aufgelöst wurden, um intersubjektive Nachvollziehbarkeit sicherzustellen (siehe Abschnitt 6.2.1.2).

Ziel dieses Vorgehens ist es, einen differenzierten Einblick in die Art und Qualität der von LLMs generierten Rückmeldungen unter kontextarmen Bedingungen zu gewinnen. Dadurch sollen sowohl Potenziale für den didaktischen Einsatz sichtbar gemacht als auch zu erwartende Herausforderungen – etwa Inkonsistenzen oder irreführende Informationen – systematisch identifiziert werden.

6.2.1.1 Erhebung LLM-generierter Rückmeldungen

Da zum Zeitpunkt der Untersuchung keine öffentlich verfügbaren Datensätze mit LLM-generierten Rückmeldungen zu studentischen Programmierlösungen vorlagen, wurde für die nachfolgend beschriebenen Untersuchungen ein eigener Datensatz erstellt. Die Erhebung gliederte sich in folgende fünf Schritte, die im Rahmen der nächsten Abschnitte näher erläutert werden:

S1 Auswahl geeigneter Programmieraufgaben aus dem Korpus studentischer Einreichungen mit typischen Fehlermustern des Anfangsunterrichts,

S2 Festlegung konkreter Submissions aus den Datensätzen,

S3 Festlegung des Modells und der Interaktionsparameter für die Generierung.

S4 Festlegung des Prompts für die Interaktion mit dem Modell.

S5 Generierung von Rückmeldungen.

Die detaillierte Beschreibung der einzelnen Schritte bis zur Materialbasis folgt dabei dem Anspruch von Mayring (2015a), die Entstehung des Analysematerials transparent zu machen. Auf diese Weise soll gewährleistet werden, dass Auswahlentscheidungen und Erhebungsvorgänge nachvollziehbar dokumentiert sind und die qualitative Analyse auf einer klar umrissenen, methodisch begründeten Datenbasis aufsetzt.

Gemäß den Prinzipien der strukturierenden Inhaltsanalyse nach Mayring (2015a) stellt die Auswahl des Ausgangsmaterials (**S1**) einen zentralen methodischen Schritt dar. Im vorliegenden Fall diente ein vollständiger Datensatz studentischer Submissions aus einem groß angelegten Einführungskurs in die Programmierung mit `Java` als Ausgangsbasis. Dieser Datensatz umfasste Einreichungen zu allen wöchentlichen Übungsaufgaben des Kurses, sodass eine Auswahl erforderlich war, um ein handhabbares Set von Aufgaben mit zugehörigen Submissions für die Untersuchung zu erhalten.

Wesentlich ist dabei die Unterscheidung zwischen Ausgangs- und Analysematerial: Die Submissions selbst werden nicht direkt kodiert, sondern dienen ausschließlich als Grundlage für die Generierung von Rückmeldungen durch das Sprachmodell. Erst die Rückmeldungen des Sprachmodells bilden die eigentliche Materialbasis für die qualitative Auswertung im Rahmen der Untersuchung.

Die Auswahl der Aufgaben mit den zugehörigen Programmierlösungen erfolgte anhand folgender Kriterien:

- Die Aufgaben stammen aus den ersten Wochen der Lehrveranstaltung und adressieren zentrale Basiskonzepte (u. a. Methoden, Bedingungen, Wiederholungen, einfache Arithmetik).
- Jede Aufgabe ist in sich abgeschlossen, behandelt eine klar abgegrenzte Problemstellung und weist keine Abhängigkeiten zu weiteren Teilaufgaben auf.
- Zu jeder Aufgabe liegen mehrere fehlerhafte Studierendenlösungen vor, die eine Bandbreite gängiger Fehlermuster (Syntaxfehler, logische Fehler, stilistische Schwächen) repräsentieren.

- Der Umfang der Programme beträgt maximal etwa 15 Zeilen Code, um eine fokussierte Rückmeldungserzeugung zu ermöglichen.
- Für jede Aufgabe stehen umfangreiche Kontextinformationen (z. B. Aufgabenbeschreibung, Musterlösung, Klassenskelett) zur Verfügung, die in der zweiten Untersuchung berücksichtigt werden können.

Auf dieser Grundlage wurden vier Aufgaben ausgewählt, die in Tabelle 6.1 dargestellt sind. Neben zwei Varianten zur Klassifikation von Dreiecken (TTBS und TTBA) umfasst das Set eine Aufgabe zur Berechnung thermodynamischer Gleichungen (TEOS) sowie eine Aufgabe zur Berechnung negativer Fibonacci-Zahlen (NEGF). Für alle Aufgaben außer NEGF erhielten die Studierenden eine Vorlage mit einem Klassenskelett, das leere Methodenrümpfe enthielt. Bei den Dreiecksaufgaben wurden ursprünglich mehrere Teilaufgaben gemeinsam gestellt. Für die hier verwendete Fassung wurden diese getrennt, sodass zwei eigenständige Aufgaben entstanden. Da beide Teilaufgaben unabhängig voneinander bearbeitet werden konnten, ermöglichte die Aufteilung eine klarere Trennung der Lösungsansätze und adressierte unterschiedliche konzeptuelle Schwerpunkte.

Aufgabe	Beschreibung	Konzepte
TEOS	Implementieren von Methoden zur Berechnung thermodynamischer Zustandsgleichungen unter Verwendung gegebener Konstanten.	Berechnungen, Klassen, Methoden, Konstanten
TTBS	Klassifizieren eines Dreiecks anhand seiner Seitenlängen (gleichseitig, gleichschenklig oder ungleichseitig).	Methoden, Berechnungen, Verzweigungen, ENUMS
TTBA	Klassifizieren eines Dreiecks anhand seiner Innenwinkel (spitz, rechtwinklig oder stumpfwinklig).	Methoden, Berechnungen, Verzweigungen, ENUMS
NEGF	Berechnung der negativen Fibonacci-Zahlen, ohne Klassen oder Methoden der Java-Standardbibliothek zu verwenden.	Methoden, Verzweigungen, Rekursion

Tabelle 6.1: Überblick der ausgewählten Programmieraufgaben aus dem Datensatz

Im Anschluss an die Aufgabenwahl wurde eine systematische Analyse der zugehörigen Studierendenlösungen durchgeführt, um typische Fehlermuster zu identifizieren und darauf basierend konkrete Submissions aus den Datensätzen auszuwählen (**S2**). Grundlage dafür war die Auswertung automatisierter Unit-Tests, die im Rahmen des Kurses für alle Einreichungen verwendet wurden. Für jede der vier Aufgaben wurden sämtliche Submissions aggregiert und auf Basis der Testergebnisse in Fehlertypen gruppiert. Als häufige Fehler galten solche, die von mindestens 5 % der Studierenden begangen wurden.

Auf dieser Grundlage konnten pro Aufgabe mehrere Fehlertypen herausgearbeitet werden, darunter fehlerhafte Bedingungen, falsche Rückgabewerte, unvollständige Fallunterscheidungen oder Verstöße gegen Aufgabenrestriktionen (z. B. unerlaubte Imports). Aus jeder Fehlergruppe wurde maximal eine konkrete Submission ausgewählt, sodass ein möglichst breites Spektrum an Fehlermustern abgedeckt war. Die Auswahl erfolgte manuell, wobei unterschiedliche strukturelle Probleme, Codeformate und Bearbeitungsstrategien berücksichtigt wurden. Zudem wurde darauf geachtet, dass die ausgewählten Lösungen für sich genommen verständlich und in sich abgeschlossen waren, ohne auf externe Dateien oder Vorbedingungen zu verweisen. Um die Qualität der Auswahl zu sichern, führten drei Personen unabhängig voneinander zunächst eine Vorauswahl

durch. Anschließend wurde im Konsens entschieden, welches finale Set an Submissions als Ausgangsbasis für die Generierung der Rückmeldungen genutzt wurde. Insgesamt wurden auf diese Weise **33 Submissions** ausgewählt, die eine vielfältige Basis für die nachfolgende Interaktion mit dem Sprachmodell darstellten – jeweils 10 für TEOS, TTBS, TTBA und 13 für NEGF.

Im nächsten Schritt wurde ein geeignetes Sprachmodell bestimmt, das zur Generierung der Rückmeldungen auf dem zuvor ausgewählten Set studentischer Lösungen eingesetzt werden sollte (**S3**). Hierfür wurde das Sprachmodell `GPT-3.5` von OpenAI (Stand: März 2023) gewählt. Die Entscheidung für dieses Modell beruhte auf zwei zentralen Überlegungen: Erstens handelte es sich um das zu diesem Zeitpunkt am weitesten verbreitete LLM, das über die ChatGPT-Webanwendung in einer frei zugänglichen Basisversion genutzt werden konnte, auch wenn es proprietär von OpenAI betrieben wurde. Zweitens war aus dem Umfeld der Lehrveranstaltung bereits bekannt, dass Studierende dieses Modell aktiv zur Unterstützung bei Programmieraufgaben einsetzten. Dieses Bild deckt sich mit entsprechenden Diskussionen in öffentlichen Foren sowie mit frühen Erfahrungsberichten zum Einsatz von ChatGPT im Hochschulkontext. Damit bot die Nutzung von `GPT-3.5` die höchste Wahrscheinlichkeit, ein reales Nutzungsszenario von Studierenden im universitären Kontext abzubilden.

Für die Interaktion mit dem Modell standen grundsätzlich zwei Zugänge zur Verfügung: die Nutzung der **API** oder die Verwendung der frei zugänglichen **ChatGPT-Webanwendung**. Während die API eine präzisere Steuerung der Modellparameter erlaubt hätte, wurde für die vorliegende Untersuchung die Nutzung der Webanwendung gewählt. Diese Entscheidung ist darauf begründet, dass die Rückmeldungen in einem Format generiert werden sollen, das einer (zum Zeitpunkt der Erhebung) typischen Nutzung durch Studierende entspricht. Da Lernende in realen Lehr-Lern-Situationen üblicherweise die Chat-Oberfläche verwenden, gewährleistet die Wahl dieses Zugangs eine hohe Authentizität der erhobenen Daten. Darüber hinaus stellte die Nutzung der offiziellen Webanwendung sicher, dass alle systemseitigen Voreinstellungen – einschließlich der nicht offengelegten Parameter wie Temperatur oder Systemprompt – dem Standardverhalten des Modells in regulären Nutzungskontexten entsprachen. Auf diese Weise konnte ausgeschlossen werden, dass die Rückmeldungen durch abweichende API-Parameter oder experimentelle Konfigurationen beeinflusst wurden.

Nach der Auswahl des Sprachmodells wurde im nächsten Schritt die Gestaltung der Eingabeaufforderungen festgelegt (**S4**). Ziel war es, ein einheitliches und möglichst neutrales Interaktionsformat zu schaffen, um Unterschiede in den Rückmeldungen möglichst eindeutig auf das Sprachmodell selbst zurückführen zu können. Als Prompt wurde für jede studentische Lösung dieselbe generische Hilfsanfrage "What's wrong with my code?" verwendet, gefolgt vom jeweiligen Quelltext der Submission. Weitere Kontextinformationen – etwa die Aufgabenstellung oder andere individuelle bzw. situative Faktoren – wurden bewusst nicht integriert, um ein kontextarmes Setting zu schaffen, das als Vergleichsbasis für die Analyse im Rahmen von **FF3.1** dient.

Auf Grundlage der festgelegten Promptgestaltung wurden anschließend die eigentlichen Modellinteraktionen durchgeführt und die Rückmeldungen generiert (**S5**). Um auch die Variabilität des Modells bei identischer Eingabe zu erfassen, wurden **für jede Submission drei Rückmeldungen generiert**. Dies erfolgte jeweils über die *Regenerate*-Funktion innerhalb desselben Chatverlaufs – auf demselben Rechner, im selben Browser und in einem konsistenten Sitzungszustand. Dieses Vorgehen soll typische Nutzungsszenarien durch Lernende widerspiegeln, die ebenfalls mehrere Modellantworten auf dieselbe Eingabe abrufen könnten, wenn sie beispielsweise mit der ersten Rückmeldung unzufrieden sind oder alternative Erklärungen wünschen.

Für jede Submission wurde ein **separater Chat** verwendet, um eine Vermischung des Kontextverlaufs zu vermeiden. Da das Sprachmodell bei der Interaktion über die ChatGPT-Oberfläche den bisherigen Dialog als impliziten Prompt berücksichtigt, hätte die Nutzung desselben Chats

zu unerwünschten Kontextübernahmen zwischen verschiedenen Aufgaben geführt. Innerhalb eines einzelnen Chats wurden daher ausschließlich die drei Varianten derselben Rückmeldung erzeugt.

Aus den 33 ausgewählten Studierendenlösungen wurden durch jeweils drei Generierungen insgesamt **99 Rückmeldungen** des Sprachmodells erhoben. Dieses Korpus stellt die eigentliche Materialbasis dar, die im Rahmen der strukturierenden Inhaltsanalyse nach Mayring (2015a) ausgewertet wird. Während die Studierendenlösungen lediglich als Ausgangspunkt dienten, bilden erst die auf dieser Grundlage erzeugten Rückmeldungen das empirische Analysematerial für die qualitative Untersuchung hinsichtlich ihrer inhaltlichen und formalen Merkmale.

6.2.1.2 Qualitative Inhaltsanalyse

Zur Erfassung der Charakteristika und der Qualität LLM-generierter Rückmeldungen (**FF3.1**) wurde ein theoriegeleitetes Kategoriensystem entwickelt, das den Prinzipien der *strukturierenden Inhaltsanalyse* nach Mayring (2015a) folgt. Grundlage hierfür bilden mehrere theoretische und empirische Quellen:

Erstens stützt sich das System auf die systematische Übersicht von Keuning, Jeuring und Heeren (2019), die verschiedene Feedbacktypen in der Programmierausbildung klassifiziert. Diese Kategorien wurden bereits in der Untersuchung zu Lehrpersonen-Feedback (Kapitel 4) eingesetzt. Ihre erneute Verwendung gewährleistet einerseits die Anschlussfähigkeit an eine in der Fachdidaktik etablierte Typologie und ermöglicht andererseits eine direkte Vergleichbarkeit zwischen menschlich erzeugtem und modellgeneriertem Feedback. Dabei wurde nicht die vollständige Typologie von Keuning, Jeuring und Heeren (2019) in ihrer Differenziertheit übernommen, sondern es wurden gezielt Teilaspekte ausgewählt und zusammengefasst, die für den Untersuchungsfokus besonders relevant waren. Die Auswahl und Reduktion der Feedbacktypen folgte drei zentralen methodischen Überlegungen:

- Der Fokus dieser Studie lag auf der Analyse von Rückmeldungen zu fehlerhaften studentischen Lösungen. Entsprechend standen Rückmeldungstypen im Vordergrund, die mit der Erklärung, Korrektur oder Veranschaulichung von Fehlern verbunden sind. Kategorien, die sich auf korrektes oder affirmatives Feedback beziehen (z. B. Bestätigung bei vollständig richtigen Lösungen), wurden nicht berücksichtigt.
- Darüber hinaus zielte das Studiendesign auf eine realitätsnahe Beschreibung von LLM-generierten Rückmeldungen unter minimalem Prompt-Kontext – in Anlehnung an typische Situationen, wie sie etwa in Foren wie StackOverflow vorkommen. In solchen Kontexten steht meist die unmittelbare funktionale Fehlerbehebung im Zentrum, während didaktisch strukturierte Unterstützung zur Förderung langfristiger Lernprozesse seltener ist. Für die Analyse war daher besonders relevant, ob Rückmeldungen über reine Lösungsvorschläge hinaus auch Hinweise auf Fehlerursachen oder konzeptuelle Missverständnisse enthielten.
- Schließlich wurden einzelne Subkategorien der Originaltypologie gezielt zusammengeführt, um die Anwendbarkeit des Kategoriensystems zu erhöhen. So unterscheidet Keuning, Jeuring und Heeren (2019) innerhalb des Typs *Knowledge on Mistakes* (`KM`) mehrere Unterformen, darunter Hinweise auf Syntaxfehler (`KM-CE`), Performance-Probleme (`KM-PI`), stilistische Schwächen (`KM-SI`) oder semantische Fehler (`KM-SE`). Diese Differenzierung wurde für die vorliegende Analyse nicht aufrechterhalten, da der Fokus nicht auf der Art des Fehlers, sondern auf der grundsätzlichen Frage lag, ob Rückmeldungen überhaupt Informationen über die Ursache eines Problems enthielten. Aus diesem Grund wurde auch der Feedbacktyp *Knowledge about Task Constraints* (`KTC`) in diese Kategorie integriert, sofern sich Rückmeldungen auf Anforderungen oder Einschränkungen der Aufgabe bezogen, die als Ursache für den Fehler betrachtet werden konnten.

Zweitens wurden zusätzliche Dimensionen berücksichtigt, die sich in der Analyse des Feedbacks durch Lehrpersonen (Kapitel 4) als relevant herausgestellt haben, in bestehenden Typologien jedoch kaum Beachtung finden. Dazu gehören insbesondere Unsicherheitsmarkierungen (`UNC`), gezielte Nachfragen zu weiterem Kontext sowie motivationale Rückmeldungen (`MOT`). Ihre Integration stellt sicher, dass auch diese in der Praxis bedeutsamen Aspekte systematisch erfasst und mit den Ergebnissen der Lehrpersonen-Studie in Beziehung gesetzt werden können.

Drittens wurden Kategorien ergänzt, die sich auf die fachliche Qualität der generierten Rückmeldungen beziehen. In der Feedbackforschung gelten die Korrektheit der gegebenen Information sowie deren praktische Verwendbarkeit als zentrale Qualitätskriterien (Shute 2008; Hattie 2009; Narciss 2007). Für Rückmeldungen in der Programmierausbildung bedeutet dies insbesondere, dass vorgeschlagene Codebeispiele syntaktisch korrekt und kompilierbar sein sollten (`COMP`) und dass die enthaltenen Informationen fachlich zutreffend sind (`MIS`). Die Aufnahme dieser Kategorien trägt dem Umstand Rechnung, dass Feedback für Lernende nur dann einen Nutzen entfalten kann, wenn es verlässlich, korrekt und unmittelbar auf die Aufgabenbearbeitung anwendbar ist.

Das resultierende Kategoriensystem vereinfacht damit die theoretische Typologie zugunsten einer empirisch praktikablen, textbasierten Anwendung. Die konkrete Zuordnung der verwendeten Analyse-Kategorien zu den theoretischen Rückmeldetypen ist in Tabelle 6.2 dokumentiert. Die Umbenennung der Kategorien (z. B. `CAUSE` statt `KM-CE`) erfolgte zudem mit dem Ziel, die Bezeichnungen stärker an der sprachlichen Realisierung in den generierten Rückmeldungen auszurichten. Da im Kategoriensystem nicht nur verschiedene Subtypen, sondern vereinzelt auch unterschiedliche Haupttypen der Keuning-Typologie (z. B. `KM` und `KTC`) innerhalb einer Analyse-Kategorie zusammengefasst wurden, erschien eine terminologische Neubenennung sinnvoll, um die theoretische Herkunft von der empirischen Operationalisierung klar abzugrenzen.

Das finale Kategoriensystem umfasst **elf Kategorien**, die den drei übergeordneten Bereichen `CONTENT`, `QUALITY` und `OTHER` zugeordnet sind. Für jede Kategorie wurde ein geschlossenes Kodierschema (z. B. Ja/Nein, Skala, Snippet/Nicht zutreffend) definiert. Eine Übersicht über die Kategorien und ihre Zuordnung zu den zugrundeliegenden Feedbacktypen aus der Literatur gibt Tabelle 6.2.

Alle **99** generierten Rückmeldungen wurden mit dem beschriebenen Kategoriensystem kodiert. Bei **11** Kategorien ergibt sich damit eine Gesamtzahl von **1089** zu treffenden Kodierentscheidungen (Presence/Absence bzw. Kategorien-spezifische Ausprägung je Rückmeldung). Die *Analyseeinheit* war jeweils die vollständige Rückmeldung (Text der Modellantwort). Die Kategorien wurden unabhängig voneinander beurteilt.

Die Kodierung erfolgte durch **zwei unabhängige Personen**, die *den gesamten Korpus* vollständig parallel kodierten (keine Aufteilung des Materials). Abweichungen wurden anschließend in einem **konsensualen Verfahren** aufgelöst, um eine einheitliche Anwendung der Kodierregeln sicherzustellen. Vorgehen, Entscheidungslogik und Ablaufschritte der Konsensbildung entsprachen dabei exakt dem in Kapitel 4 beschriebenen Prozess und werden hier aus Gründen der Redundanz nicht erneut ausführlich dargestellt.

6.2.2 Ergebnisse

Die Ergebnisse der Auswertung werden in erster Linie *deskriptiv-qualitativ* berichtet. Ergänzend werden an einigen Stellen *quantitative Häufigkeiten* angegeben, um Tendenzen und Relativgewichte der Kategorien transparent zu machen. Eine solche Quantifizierung einzelner Kodierergebnisse ist im Rahmen der strukturierenden Inhaltsanalyse zulässig, sofern sie der inhaltlichen Beschreibung dient und nicht als inferenzstatistische Testung missverstanden wird (Mayring

Kategorie	Beschreibung	Quelle
CONTENT		
INFO	Nachfrage nach weiteren Informationen oder Kontext	: GETI
STYLE	Stilistische Verbesserungsvorschläge	: KM-SI
CAUSE	Fokus auf Fehlerursache	: KM-CE, KM-SE, KM-PI, KTC
FIX	Fokus auf Fehlerbehebung	: KH-EC
CODE	Bereitstellung von Quelltext	: KH-EC, KCR
EXA	Veranschaulichung durch Beispiele	: KC-EXP, KC-EXA
QUALITY		
COMP	Angegebener Code ist syntaktisch korrekt/kompilierbar	⊡
MIS	Rückmeldung enthält inhaltlich irreführende Aussagen	⊡
UNC	Ausdruck von Unsicherheit oder Unklarheit	⌘: UNC
OTHER		
META	Metakognitive Hinweise (z. B. Teststrategien, Selbstreflexion)	: KMC
MOT	Motivationale Elemente (z. B. Ermutigung, positive Rückmeldung)	: MOT

: siehe Tabelle 4.12 ⌘: siehe Tabelle 4.11 ⊡: allg. Kriterien an Qualität (Shute 2008; Hattie 2009)

Tabelle 6.2: Übersicht verwendeter Kodierkategorien mit theoretischer und empirischer Grundlage

2015a). Mit dieser Kombination adressieren die Ergebnisse **FF3.1** in beiden Teilen: *Charakteristika* der Rückmeldungen (inhaltlich) und *Qualitätsaspekte* (z. B. irreführende Inhalte, Unsicherheitsmarkierungen). Tabelle 6.3 zeigt die Auftretenshäufigkeiten aller elf Kategorien über drei Generierungen (R1-R3) für jede der 33 Submissions.

(1) Variabilität der Modellantworten bei identischem Input: Über alle Submissions und Kategorien hinweg (33 Submissions × 11 Kategorien = 363 Kategorie-Triplets) zeigten sich in **116** Fällen Unterschiede zwischen den einzelnen Generierungen (≈ 32 %). Auf Ebene der Submissions variierte in **27 %** der Fälle mindestens ein Merkmal zwischen den drei Antworten. Damit sind die generierten Rückmeldungen unter Minimalkontext nur eingeschränkt reproduzierbar und können inhaltlich spürbar schwanken. Dieses Befundmuster ist für **FF3.1** bedeutsam, weil es zeigt, dass die *Charakteristika* der Rückmeldungen ohne Kontext nicht stabil sind.

(2) Inhaltliche Muster (CONTENT): Am häufigsten traten Rückmeldungen auf, die **Fehlerursachen erläutern** (CAUSE: 88/99) und **konkrete Schritte zur Behebung** beschreiben (FIX: 83/99). In **65/99** Fällen stellte das Modell vollständigen Code bereit (CODE); in weiteren **13** Fällen wurden Codeausschnitte/Beispielcode geliefert. **Beispiele/Veranschaulichungen** (EXA) waren seltener und meist knapp bzw. implizit realisiert. **Stilhinweise** (STYLE: 30/99) bezogen sich überwiegend auf Benennungen oder Formatierung. **Nachfragen nach weiterem Kontext** (INFO) wurden nur in **4/99** Rückmeldungen explizit formuliert. In Summe deutet das auf ein primär *problem- und lösungsorientiertes* Antworten unter Minimalprompt hin (CAUSE/FIX/CODE), mit vergleichsweise wenig metakommunikativer Kontextklärung (INFO). Auch dies ist hinsichtlich der Beantwortung von **FF3.1** relevant: Ohne bereitgestellten Kontext fragt das Modell kaum aktiv nach – stattdessen werden vor allem Erklärungen, Korrekturen und Code *produziert*.

(3) Qualitätsbezogene Muster (QUALITY): Ein zentrales Ergebnis betrifft **irreführende oder sachlich unzutreffende Aussagen** (MIS): In **61/99** Rückmeldungen waren entsprechende Elemente enthalten. Bezogen auf die Aufgabenebene zeigten sich in **24/33** Submissions in mindestens einer der drei Antworten Vorkommnisse fehlleitender Informationen (MIS). Dies

Tabelle 6.3: Ergebnisse der Kodierung der generierten Rückmeldungen mit minimalem Kontext

	CONTENT																QUALITY									OTHER					
	INFO			**STYLE**			**CAUSE**			**FIX**			**CODE**			**EXA**			**COMP**			**MIS**			**UNC**			**META**			**MOT**		
ID	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
01_TEOS	○	○	○	○	○	○	○	●	●	○	○	○	●	◐	●	○	○	○	●	–	●	○	●	●	○	○	○	○	○	○	○	○	○
02_TEOS	○	○	○	●	●	●	●	●	●	●	●	●	●	●	●	○	○	○	●	●	●	●	●	○	○	○	○	○	○	○	○	○	○
03_TEOS	○	●	○	○	○	○	●	●	●	●	●	●	◐	○	○	○	○	○	–	–	–	●	○	○	○	○	○	○	○	○	○	○	○
04_TEOS	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	–	–	–	○	○	○	●	●	●	●	●	●	○	○	○
05_TEOS	○	○	○	●	●	●	●	●	●	●	●	●	●	●	●	○	○	○	●	●	●	○	○	○	○	○	○	○	○	○	○	○	○
06_TEOS	○	○	○	●	○	●	●	●	●	●	●	●	◐	●	◐	●	●	●	–	●	–	○	○	○	○	○	○	○	○	○	○	○	○
07_TEOS	○	○	○	○	○	○	●	●	●	●	●	●	●	●	○	○	○	○	●	●	–	○	●	○	○	○	○	○	○	○	○	○	○
08_TEOS	○	○	○	○	●	○	●	○	●	●	●	●	◐	○	○	○	○	○	–	–	–	○	○	○	○	○	○	○	○	○	●	○	○
09_TEOS	○	○	○	○	●	○	●	●	●	●	●	●	●	●	●	○	○	○	●	●	●	●	●	●	○	○	○	○	○	○	○	○	○
10_TEOS	○	○	○	○	●	○	●	●	●	●	●	●	●	●	●	○	○	○	○	●	●	●	○	○	○	○	○	○	○	○	○	○	○
01_TTBS	○	○	○	○	○	○	●	●	●	○	●	●	○	●	●	○	●	○	–	●	●	●	○	○	●	○	○	○	○	○	○	○	○
02_TTBS	○	○	○	○	○	○	●	●	●	●	●	●	●	●	●	○	○	●	–	●	●	○	○	○	○	○	○	○	○	○	○	○	○
03_TTBS	○	○	●	○	●	●	●	○	●	●	○	○	●	○	○	○	○	○	●	–	–	●	○	●	○	○	●	○	○	○	○	○	○
04_TTBS	○	○	○	○	○	○	●	●	●	●	●	●	●	◐	●	○	○	○	●	●	–	○	○	●	○	○	○	○	○	○	○	○	○
05_TTBA	○	○	○	○	○	●	●	●	●	●	●	●	◐	●	●	○	●	○	–	●	●	○	●	●	○	○	○	○	○	○	○	○	○
06_TTBA	○	○	○	●	○	○	●	●	●	●	●	●	○	●	◐	○	○	○	–	●	–	●	●	●	●	○	○	○	○	○	○	○	○
07_TTBA	○	○	○	●	●	○	●	●	●	●	●	●	○	◐	◐	○	○	●	–	–	–	●	●	●	○	●	○	○	○	○	○	○	○
08_TTBA	○	○	○	○	○	○	●	●	●	●	●	●	●	●	●	●	○	●	●	○	●	●	●	●	○	○	○	○	○	○	○	○	○
09_TTBA	○	○	○	○	○	○	●	○	●	●	●	●	●	●	●	○	○	○	●	●	●	●	○	●	○	○	○	○	○	○	○	○	○
10_TTBA	○	○	○	○	○	○	●	●	●	●	●	●	●	●	●	●	○	○	●	●	●	●	●	●	○	○	○	○	○	○	○	○	○
01_NEGF	○	○	○	○	●	●	●	○	●	●	○	●	●	○	○	●	○	○	●	–	–	●	○	●	○	●	●	○	●	○	○	○	○
02_NEGF	○	○	○	○	○	●	●	●	●	●	●	●	●	●	○	○	○	○	●	●	–	●	●	●	○	○	●	○	○	○	○	○	●
03_NEGF	○	○	○	○	○	●	●	●	●	●	●	●	●	●	●	○	○	○	●	●	●	●	●	●	○	○	○	○	○	○	●	○	○
04_NEGF	●	○	○	○	○	●	●	●	○	○	●	○	○	●	●	○	○	○	–	●	●	○	●	●	●	○	●	○	○	○	○	○	○
05_NEGF	○	●	○	○	○	○	●	○	○	●	○	○	◐	○	○	○	○	○	–	–	–	○	○	○	●	●	●	○	○	○	○	○	○
06_NEGF	○	○	○	●	●	●	●	●	●	●	○	●	●	○	○	○	○	○	●	–	–	●	○	○	●	●	●	○	○	○	○	○	○
07_NEGF	○	○	○	○	○	○	●	●	●	●	●	●	●	◐	●	○	○	○	–	–	●	●	●	●	○	○	○	○	○	○	○	○	○
08_NEGF	○	○	○	●	○	○	●	●	●	●	●	●	●	●	●	○	○	○	●	–	●	●	●	●	○	○	○	○	○	○	●	○	○
09_NEGF	○	○	○	●	●	○	●	●	●	●	●	●	●	●	●	○	○	●	○	○	○	●	●	●	○	○	○	○	○	○	○	○	○
10_NEGF	○	○	○	●	○	○	●	●	●	●	●	●	●	●	●	○	○	○	●	●	●	●	○	●	○	○	○	○	○	○	○	○	○
11_NEGF	○	○	○	○	○	○	●	●	●	○	●	●	●	●	●	○	○	○	○	○	●	●	●	●	○	○	○	○	○	○	○	○	○
12_NEGF	○	○	○	○	○	○	●	●	●	●	●	●	●	●	●	○	○	○	●	●	●	●	●	●	●	○	○	○	○	○	○	○	○
13_NEGF	○	○	○	○	○	○	●	●	●	●	●	●	◐	●	●	○	○	○	–	●	○	●	●	●	○	●	●	○	○	○	○	○	○

●: Ja ○: Nein ◐: Snippet –: Nicht anwendbar

reichte von falschen Korrekturvorschlägen über missverständliche Erklärungen bis zu inhaltlich irrelevanten Hinweisen. Dabei sei angemerkt, dass die Kategorie `MIS` nicht direkt fachliche Fehler bedeutet, sondern auch Hinweise zu Lösungsstrategien, die inhaltlich nicht zielführend sind oder im Kontext der Lehrveranstaltung nicht erlaubt (wie z. B. die Nutzung bestimmter Bibliotheken). **Unsicherheitsmarkierungen** (`UNC`) wurden in **21/99** Antworten identifiziert; **14** davon entfielen auf die komplexere Aufgabe `NEGF`. Typische Formulierungen lauteten etwa: *„Auf Basis des bereitgestellten Codes ist es schwierig, den Fehler eindeutig zu bestimmen.“* Diese Befunde adressieren den Qualitätsaspekt von **FF3.1**: Unter kontextarmen Bedingungen treten *Unsicherheit* und insbesondere *irreführende Inhalte* häufig auf – mit potenziellen Konsequenzen für die Nutzbarkeit des Feedbacks.

(4) Metakognitive und motivationale Elemente (OTHER): Metakognitive Hinweise (`META`) wurden nur vereinzelt gefunden (z. B. *„Teste Zwischenergebnisse mit Ausgaben.“*); **motivationale Elemente** (`MOT`) traten ebenfalls selten auf (etwa knappe Ermutigungen wie *„Hope this helps.“*).

(5) Aufgabenspezifische Beobachtungen und Typologienbezug: Der Typ *Knowledge about Task Constraints* (`KTC`) wurde erwartungsgemäß nicht beobachtet, da die Aufgabenstellungen im Minimalprompt nicht bereitgestellt wurden. Umgekehrt zeigte sich sehr häufig eine Komponente, die dem Typ *Knowledge of the Correct Response* (`KCR`) entspricht: Zahlreiche Antworten enthielten vollständige korrigierte Lösungen. Obwohl `KCR` in der Typologie von Keuning, Jeuring und Heeren (2019) nicht als elaboriert gilt, ist er für LLM-Feedback unter Minimalbedingungen offenbar charakteristisch (direkte Code-Substitution als Standardstrategie).

6.2.3 Zwischenfazit: Generierung von Feedback ohne Kontext

Die Untersuchung zeigt, dass große Sprachmodelle auch unter minimalen Eingabebedingungen in der Lage sind, vielfältige Rückmeldungsformen zu generieren, die sowohl mögliche Ursachen für Fehler als auch konkrete Korrekturanleitungen und Beispielcode umfassen. Damit unterscheiden sich LLM-basierte Rückmeldungen qualitativ von klassischen, regelbasierten Systemen und eröffnen neue Perspektiven für individualisierte Unterstützung im Programmierlernen. Gleichzeitig wird deutlich, dass die Qualität dieser Rückmeldungen ohne Kontext erheblichen Schwankungen unterliegt. Die Analysen zeigen (a) eine starke Ausrichtung auf erklärende und korrigierende Elemente (`CAUSE`, `FIX`, `CODE`), (b) eine geringe Tendenz zur Kontextklärung (`INFO`), (c) häufiges Auftreten irreführender oder unzutreffender Inhalte (`MIS`) sowie (d) nur seltene metakognitive oder motivationale Anteile (`META`, `MOT`). Auch die Variabilität der Antworten bei identischem Input unterstreicht die begrenzte Stabilität der generierten Rückmeldungen.

Insgesamt lässt sich festhalten, dass LLMs unter kontextarmen Bedingungen vor allem problem- und lösungsorientierte Rückmeldungen erzeugen, dabei jedoch meist unsystematisch und teils fehlerhaft vorgehen. Für den didaktischen Einsatz bedeutet dies, dass solche Rückmeldungen zwar Potenzial für spontane Unterstützung bieten, aber nur eingeschränkt als verlässliche oder qualitativ kontrollierte Feedbackquelle dienen. Die Ergebnisse legen nahe, dass eine gezielte Kontextualisierung der Prompteingabe nötig ist, um Qualität, Passung und Steuerbarkeit der Rückmeldungen zu verbessern, was in nachfolgender Untersuchung systematisch geprüft wird.

6.3 Generierung von Rückmeldungen mit Aufgaben- und Lernkontext

Die in Abschnitt 6.2 dargestellten Ergebnisse zeigen, dass große Sprachmodelle auch unter Minimalprompts elaborierte Rückmeldungen erzeugen können, diese jedoch mit ausgeprägter Varianz, teils irreführenden Informationen und wenigen metakognitiven bzw. motivierenden Elementen einhergehen. Vor diesem Hintergrund untersucht die zweite Studie, ob und wie sich *Qualität*, *Adressatengerechtigkeit* und *Typenspezifik* der Rückmeldungen durch eine systematische *Kontextualisierung* der Prompteingabe verbessern und gezielt steuern lassen. Methodisch knüpft die Untersuchung an den in Kapitel 5 entwickelten formalen Bezugsrahmen an und ist im Design Science Research-Prozess (vgl. Abschnitt 3.3.1) dem *Design Cycle* zuzuordnen: Das *Artefakt* ist ein *Rückmeldungsgenerator*, der aus Aufgaben- und Antwortkontext sowie einer expliziten Typinstruktion Rückmeldungen erzeugt. Der Prompt wird iterativ so lange angepasst, bis der Generator mit hinreichender Zuverlässigkeit Rückmeldungen des geforderten Typs produziert.

Ziel der zweiten Untersuchung ist es, systematisch zu erheben, inwiefern (a) die *Qualität* LLM-generierter Rückmeldungen durch kontextreiches Prompting steigt (u. a. Reduktion irreführender Inhalte, Präzisierung der Bezüge) und (b) *spezifische Feedbacktypen* im Sinne etablierter Taxonomien (Keuning, Jeuring und Heeren 2019) gezielt und verlässlich generiert werden können. Die Studie operationalisiert damit insbesondere **FF3.2** (Steuerbarkeit von Feedbacktypen) und ergänzt die in Studie 1 adressierte Qualitätsdimension von **FF3.1** um den Faktor *Kontext*.

6.3.1 Methodisches Vorgehen

Das methodische Vorgehen der zweiten Untersuchung folgt grundlegend der Methodik der ersten Untersuchung mit Minimalkontext (vgl. Abschnitt 6.2). Kern ist erneut die Generierung und qualitative Analyse von LLM-basierten Rückmeldungen zu realen studentischen Programmierlösungen mittels einer theoriegeleiteten, deduktiven Kategorienanwendung. Um Redundanzen zu vermeiden, werden im Folgenden ausschließlich die Unterschiede und Erweiterungen gegenüber der ersten Untersuchung beschrieben. Alle weiteren methodischen Schritte (z. B. Kodierprozess, Konsensverfahren) entsprechen dem Vorgehen in Abschnitt 6.2 und werden hier nicht erneut im Detail wiederholt.

6.3.1.1 Erhebung LLM-generierter Rückmeldungen

Für die zweite Untersuchung wird der zuvor verwendete Aufgabenkorpus grundsätzlich übernommen. Eine Ausnahme bildet die ursprünglich geteilte `Triangle`-Aufgabe, die im Rahmen der ersten Untersuchung in die beiden Teilaufgaben `TTBS` (Klassifikation nach Seitenlängen) und `TTBA` (Klassifikation nach Innenwinkeln) zerlegt worden war. Diese Trennung diente dort der isolierten Analyse einzelner Methodenrümpfe und war inhaltlich unproblematisch. Für die kontextualisierte Untersuchung wird die Aufgabe jedoch wieder in ihrer ursprünglichen Form (`TTBSA`) betrachtet, da sich Aufgabenstellung und Musterlösung auf beide Teilaspekte beziehen und nur im Verbund einen vollständigen instruktionalen Kontext ergeben. Die Wiederherstellung war somit erforderlich, um eine konsistente Kontextbereitstellung im Prompt zu gewährleisten.

Darüber hinaus wird der Korpus mit dem Ziel erweitert, die Generalisierbarkeit der Befunde zu erhöhen und das Rückmeldeverhalten großer Sprachmodelle in einem breiteren Spektrum von Aufgabenarten, Fehlertypen und insbesondere Programmiersprachen zu analysieren. Die Auswahl der zusätzlichen Aufgaben folgt denselben Kriterien wie in Untersuchung 1 (vgl. Abschnitt 6.2.1.1), wurde jedoch um die gezielte Variation der Programmiersprache ergänzt. Auf diese Weise soll die Untersuchung eines möglichen Einflusses der Programmiersprache auf Qualität und Spezifik der generierten Rückmeldungen ermöglicht werden. Neben `Java` wurden daher auch Aufgaben in `Python`, `Dart`, `Kotlin` und `C++` aufgenommen.

Die zusätzlichen Aufgaben stammen aus etablierten Datensätzen, die bereits im Rahmen der Studie zu Lehrpersonen-Feedback (Kapitel 4) berücksichtigt wurden und damit inhaltlich an die dort untersuchten Feedbacksituationen anschließen. Konkret werden drei Aufgaben ergänzt (vgl. Tabelle 6.4): **PASS** (Password-Validator, FITech (Lehtinen 2022), `Dart`), **MAX3** (Maximum of Three, TaskTracker (Lyulina et al. 2021), mehrere Sprachen) und **FIB** (rekursive Fibonacci-Berechnung, CodingBat (Kiesler, Goethe-Universität Frankfurt Am Main und Hochschule Fulda 2022)).

Während in der ersten Untersuchung pro Aufgabe mehrere Submissions berücksichtigt werden konnten, musste die Zahl der Lösungen in dieser Untersuchung bewusst reduziert werden, um den mit der erweiterten Kodierung verbundenen Analyseaufwand praktikabel zu halten. Hintergrund ist, dass im Rahmen der zweiten Untersuchung nicht nur eine Rückmeldung pro Submission generiert wird, sondern gezielt unterschiedliche *Feedbacktypen* angefordert werden. Dadurch vervielfacht sich die Anzahl der zu analysierenden Rückmeldungen gegenüber der ersten Studie erheblich. Eine größere Zahl an Submissions hätte daher zu einem Volumen an zu kodierendem Material geführt, das mit den verfügbaren Ressourcen nicht mehr zuverlässig kodierbar gewesen wäre. Grundsätzlich wurde deshalb *eine Submission pro Aufgabe* ausgewählt, wobei darauf geachtet wurde, dass das Gesamtkorpus ein heterogenes Spektrum an Fehlertypen (z. B. syntaktisch, semantisch, stilistisch) abbildet.

Von diesem Vorgehen wurde in zwei Fällen gezielt abgewichen:

- Für die Aufgabe `MAX3` wurden mehrere Submissions in unterschiedlichen Programmiersprachen aufgenommen, um sprachübergreifende Vergleiche bei identischer Aufgabenstellung zu ermöglichen.
- Bei `MAX3` und `FIB` wurden zusätzlich neben fehlerhaften auch korrekte Lösungen in das Ausgangsmaterial aufgenommen. Dadurch sollte sichtbar werden, ob das Modell auch bei korrekten Lösungen Rückmeldungen generiert, die auf vermeintliche Probleme verweisen oder implizit Fehler unterstellen.

Tabelle 6.4 gibt einen Überblick über die insgesamt **6 Aufgaben** mit **11 Submissions** (darunter auch Formate aus der ersten Studie) sowie die jeweils berücksichtigten Programmiersprachen.

Name	Beschreibung	Konzepte	Sprache	Verwendung
Physics	Implementieren von Methoden zur Berechnung thermodynamischer Zustandsgleichungen unter Verwendung gegebener Konstanten.	Berechnungen, Klassen, Methoden, Konstanten	Java	Prompt-Entwicklung, Evaluation
Triangles	Klassifizieren eines Dreiecks anhand (1) seiner Seitenlängen (gleichseitig, gleichschenklig oder ungleichseitig) und (2) seiner Innenwinkel (spitz, rechtwinklig oder stumpfwinklig).	Methoden, Berechnungen, Verzweigungen, ENUMS	Java	Prompt-Entwicklung, Evaluation
NegaFib	Berechnung der negativen Fibonacci-Zahlen, ohne Klassen oder Methoden der Java-Standardbibliothek zu verwenden.	Methoden, Verzweigungen, Rekursion	Java	Prompt-Entwicklung, Evaluation
Fibonacci	Berechnung der negativen Fibonacci-Zahlen, ohne Klassen oder Methoden der Java-Standardbibliothek zu verwenden.	Methoden, Verzweigungen, Rekursion	Java	Evaluation
Password	Implementierung einer Methode, die den Nutzer so lange nach einem Passwort fragt, bis er das korrekte (als Parameter übergebene) eingibt.	Methoden, Parameter, Wiederholungen, Verzweigungen, Ein-/Ausgabe	Dart	Evaluation
MaxOf3	Ausgabe der größten von drei Zahlen, die als Parameter übergeben werden.	Parameter, Verzweigungen, Ein-/Ausgabe	Python Kotlin, Java, C++	Evaluation

Tabelle 6.4: Übersicht der Aufgaben für Prompt-Entwicklung und -Evaluation

Für die Generierung der Rückmeldungen wurde das Modell `GPT-4` von OpenAI ausgewählt (Stand: März 2024, Zugriff über die ChatGPT-Webanwendung) (OpenAI 2023). Zum Zeitpunkt der Untersuchung galt dieses Modell als eines der leistungsfähigsten öffentlich nutzbaren LLMs. Zwar war der Zugang zum Zeitpunkt der Untersuchung kostenpflichtig, jedoch zeichnete sich bereits ab, dass institutionelle Lizenzen, Bildungskooperationen oder Modellintegrationen (z. B. in frei verfügbare Systeme) den Zugang künftig erleichtern würden. Die gewählte Modellversion erscheint daher auch aus retroperspektiver Sicht plausibel.

Der Einsatz von `GPT-4` stellt im Vergleich zur ersten Untersuchung – in der `GPT-3.5` verwendet wurde – eine methodische Abweichung dar, die die unmittelbare Vergleichbarkeit der Ergeb-

ID	Datensatz	Aufgabe	Sprache	Fehler
Einfacher Korpus zur Entwicklung des Prompts				
10_TEOS	FIE	Physik	Java	*Syntax-Fehler* Nutzung einer nicht initialisierten Variablen Fehlender Variablentyp
13_NEGF	FIE	NegaFib	Java	*Semantische Fehler* Falscher Operator für Multiplikation Falsche Parameter im rekursiven Funktionsaufruf *Sonstiges:* Unerlaubte Bibliotheksnutzung
01_TTBSA	FIE	Dreiecke	Java	*Semantische Fehler* Fehlende Prüfungen für ungültige Dreiecke Überflüssige Prüfungen
Erweiterter Korpus für vollständige Evaluation				
FIB_01	Codingbat	Fibonacci	Java	*Semantische Fehler* Falscher rekursiver Aufruf
FIB_02	Codingbat	Fibonacci	Java	*Keine Fehler*
PASS	FITech	Password	Dart	*Semantische Fehler* Überschreiben einer Variablen Falsche Logik (fehlende Schleife) Falscher Datentyp
MAX3_01	TaskTracker	MaxOf3	Python	*Semantischer Fehler* Falsche Bedingung
MAX3_02	TaskTracker	MaxOf3	Kotlin	*Keine Fehler*
MAX3_03	TaskTracker	MaxOf3	C++	*Stilistische Probleme* Ungewöhnliche Variablennamen (durch Post-Processing) Unnötiger Import
MAX3_04	TaskTracker	MaxOf3	Java	*Keine Fehler*
MAX3_05	TaskTracker	MaxOf3	Java	*Semantische Fehler* Fehlende Bedingungen Ungewöhnlicher Ansatz mit Subtraktion in Bedingungen

Tabelle 6.5: Übersicht der ausgewählten Studierendenlösungen

nisse zur Qualität der Rückmeldungen einschränkt. Diese Entscheidung wurde jedoch bewusst getroffen: primäres Ziel der zweiten Untersuchung war nicht ein Modellvergleich, sondern die Analyse, ob sich durch kontextualisiertes Prompting gezielt bestimmte Feedbacktypen steuern lassen. Für dieses Ziel ist die Wahl eines möglichst leistungsfähigen Modells methodisch sinnvoll, u. a. da `GPT-4` auf einer breiteren und aktuelleren Datenbasis trainiert wurde und dadurch ein größeres Spektrum an domänenspezifischem Wissen (einschließlich Programmieraufgaben, Lernmaterialien und Diskussionen in Fachforen) abbildet. Ein zusätzlicher Vorteil von `GPT-4` liegt im erweiterten *Kontextfenster*, das zwar für die hier untersuchten Aufgaben nicht zwingend erforderlich gewesen wäre, jedoch die Grundlage für zukünftige Studien schafft, in denen umfangreichere Materialien (z. B. vollständige Vorlesungsskripte oder längere Bearbeitungsverläufe) als Eingabekontext berücksichtigt werden sollen. Auf diese Weise ermöglicht die Verwendung von `GPT-4`, die Befunde der vorliegenden Studie perspektivisch mit künftigen Untersuchungen in Beziehung zu setzen, ohne durch Limitierungen des Kontextumfangs eingeschränkt zu sein.

Analog zur ersten Untersuchung erfolgt die Generierung über die offizielle ChatGPT-Weboberfläche. Für jede Studierendenlösung wird ein separates Chat-Fenster verwendet, um eine Vermischung des Kontexts mit anderen Aufgaben zu verhindern und die Konsistenz des Promptkontexts sicherzustellen.

Im Gegensatz zur ersten Untersuchung, in der ein kontextarmer Prompt verwendet wurde, soll in der zweiten Untersuchung ein *kontextualisierter Prompt* zum Einsatz kommen. Dieser soll neben dem Quelltext der Studierendenlösung zusätzliche kontextuelle Informationen enthalten. Die Entwicklung erfolgte in einem mehrstufigen, iterativen Verfahren mit dem Ziel, Rückmeldungen zu erzeugen, die *typenspezifisch*, *adressatengerecht* und *inhaltlich korrekt* sind. In den folgenden Abschnitten werden die einzelnen Entwicklungsschritte näher erläutert.

Für die iterative Entwicklung des Prompts wurde ein Teilkorpus bestehend aus den drei Aufgaben der ersten Untersuchung ausgewählt: `TEOS`, `TTBSA` und `NEGF`. Diese Wahl stützte sich auf drei zentrale Gründe: Erstens lag für diese Aufgaben bereits eine vertiefte Analyseerfahrung vor, sodass bekannte Fehlermuster, Aufgabenmerkmale und Rückmeldungstypen gezielt in die Entwicklung einbezogen werden konnten. Zweitens decken die Aufgaben unterschiedliche didaktische Anforderungsbereiche des Programmieranfangsunterrichts ab – arithmetische Berechnungen (`TEOS`), bedingte Logik und Fallunterscheidungen (`TTBSA`) sowie rekursive Problemlösungen (`NEGF`). Drittens standen für diese Aufgaben vollständige Begleitmaterialien aus realen Kurskontexten zur Verfügung, was eine authentische Promptgestaltung erlaubte.

Auf die Verwendung korrekter Lösungen im Teilkorpus wurde bewusst verzichtet, da diese für die gezielte Entwicklung des Prompts nicht erforderlich waren. Stattdessen wurden korrekte Lösungen erst in der späteren Analysephase mit dem erweiterten Datensatz berücksichtigt. So ließ sich nachvollziehen, ob das Modell auch bei tatsächlich korrekten Lösungen Rückmeldungen erzeugt, die auf vermeintliche Fehler verweisen oder implizit Probleme unterstellen. Eine frühzeitige Einbeziehung solcher Lösungen hätte das Risiko mit sich gebracht, den Prompt unbeabsichtigt auf fehlerfreie Fälle zuzuschneiden und die Aussagekraft der späteren Ergebnisse zu verzerren.

Als konzeptionelle Grundlage dienten dabei sprachliche und gestalterische Prinzipien, wie sie u. a. von Denny et al. (2021) zur Gestaltung verständlicher Programmierfehlermeldungen formuliert wurden. Die folgenden Leitlinien wurden für die Gestaltung des Prompts herangezogen:

- **Prägnanz:** Die generierten Rückmeldungen sollten möglichst kurz, aber dennoch informativ sein. Diese Bedingung bezieht sich auf den Leitsatz der *Wortökonomie*, was bedeutet, dass "nur so viele Wörter verwendet werden, wie nötig sind, um den Punkt zu kommunizieren" (Denny et al. 2021).
- **Adressatenorientierung:** Die Rückmeldungen sollten sich an unerfahrene Programmieranfängerinnen und Programmieranfänger richten. Informationen über die Lernenden, wie z. B. Kompetenzprofile, Daten zu zuvor bearbeiteten Aufgaben etc. waren nicht bekannt. Diese Bedingung bezieht sich auf die Richtlinien "Jargon entfernen" und "Einfaches Vokabular verwenden".
- **Instruktivität ohne Lösungsoffenlegung:** Rückmeldungen sollten auf hilfreiche Hinweise oder Strategien abzielen, jedoch nicht direkt die vollständige Lösung preisgeben.
- **Feedback-Typ-Spezifik:** Jede Rückmeldung sollte möglichst einem der in Tabelle 2.1 beschriebenen elaborierten Feedbacktypen eindeutig zuordenbar sein.
- **Kontextintegration:** Dem Modell sollten dieselben Kontextinformationen zur Verfügung gestellt werden wie den Studierenden im Bearbeitungskontext – einschließlich Aufgabenbeschreibung, Tests und Klassenskelett. Die Musterlösung, die den Studierenden nicht zur Verfügung stand, wurde dem Modell zusätzlich bereitgestellt.

Darüber hinaus wurde eine weitere Empfehlung von Denny et al. (2021) aufgegriffen: *„Schreiben Sie Nachrichten in vollständigen Sätzen“*. Die Instruktionen innerhalb des Prompts wurden daher konsequent in vollständigen, klar strukturierten Sätzen formuliert, um dem Modell eine sprachlich kohärente und eindeutig interpretierbare Ausgangsbasis bereitzustellen. Dies sollte nicht nur die Wahrscheinlichkeit erhöhen, dass die generierten Rückmeldungen gut lesbar und adressatengerecht ausfallen, sondern auch sicherstellen, dass der Prompt für Dritte nachvollziehbar bleibt und bei Bedarf gezielt angepasst oder weiterverwendet werden kann.

Die Entwicklung des kontextualisierten Prompts folgte einem Vorgehen, das sich methodisch am *Design Cycle* des Design Science Research-Prozesses orientiert. Iterative Durchläufe mit laufender Evaluation dienen dazu, das Artefakt – den Rückmeldungsgenerator – schrittweise zu verbessern und hinsichtlich seiner Funktionalität zu validieren:

Auf Grundlage der definierten Anforderungen wurde zunächst eine erste Version des Prompts formuliert. Diese enthielt alle kontextrelevanten Materialien zu den Aufgaben (z. B. Aufgabenstellung, Klassengerüst, Zielgruppe) sowie eine explizite Instruktion zur Erzeugung eines gewünschten Feedbacktyps.

Für jede der drei Aufgabenlösungen im Teil-Datensatz wird pro Feedbacktyp eine Rückmeldung generiert, sodass in jeder Iteration insgesamt 18 Rückmeldungen entstehen (6 Feedbacktypen zu je 3 Aufgabenlösungen). Diese Rückmeldungen werden hinsichtlich der Übereinstimmung mit dem intendierten Feedbacktyp sowie ihrer inhaltlichen, sprachlichen und didaktischen Qualität analysiert. Zwei Personen kodieren hierfür die Rückmeldungen unabhängig voneinander anhand des Kategoriensystems (siehe Tabelle 4.12). Unstimmigkeiten werden konsensual aufgelöst (Hill, Thompson und Williams 1997; Hill et al. 2005). Um den Kodieraufwand bei der durch die systematische Generierung stark vergrößerten Materialbasis praktikabel zu halten, beschränkt sich die Generierung sowie Analyse dabei auf die *Hauptkategorien* der Feedbacktypologie. Eine Auswertung auf Subtypen-Ebene hätte die Zahl der zu treffenden Kodierentscheidungen erheblich gesteigert, ohne zusätzlichen Erkenntnisgewinn für die zentrale Fragestellung der Steuerbarkeit zu bieten. Auf Basis der Analyseergebnisse wurden Möglichkeiten der Überarbeitung des Prompts diskutiert und umgesetzt – insbesondere in Bezug auf Formulierungsgenauigkeit, Struktur und Typenspezifik. Dadurch entstand ein wiederkehrender Zyklus aus Generierung, Evaluation und Revision, der über mehrere Iterationen hinweg eine fortschreitende Verbesserung des Artefakts bewirkte. Der Prozess wurde beendet, wenn keine wesentlichen Verbesserungen der Rückmeldungen mehr erkennbar waren oder die generierten Ausgaben konsistent die angestrebten Qualitäts- und Typanforderungen erfüllten.

Nachdem das iterative Verfahren eine stabile und konsistente Promptfassung hervorgebracht hat, wird die finale Version des Prompts gezielt auf den zuvor *zurückgehaltenen Datensatz* angewendet. Dieser Schritt dient der *Validierung* des Artefakts: Für alle ausgewählten Aufgaben und Submissions werden typenspezifische Rückmeldungen generiert, die anschließend systematisch analysiert werden. Durch den bewussten Ausschluss dieser Daten aus der Entwicklungsphase sollte sichergestellt werden, dass die Steuerbarkeit des Rückmeldeverhaltens nicht lediglich ein Artefakt der im Entwicklungsprozess verwendeten Submissions darstellt, sondern auch bei neuen, bislang ungenutzten Aufgaben und Lösungen konsistent wirksam bleibt. Auf diese Weise wurde einer Überanpassung an den Entwicklungskorpus (*Overfitting*) methodisch vorgebeugt.

6.3.1.2 Analyse der generierten Rückmeldungen

Analog zur ersten Untersuchung werden die durch das Sprachmodell erzeugten Rückmeldungen qualitativ analysiert, um inhaltliche, qualitative und metakognitive Merkmale systematisch zu erfassen. Dabei kommt dasselbe theoriegeleitete Kategoriensystem zum Einsatz wie in Abschnitt 6.2.1.2 beschrieben. Ziel der Analyse ist erneut eine deskriptive Charakterisierung der

Rückmeldungen, nicht jedoch eine normative Qualitätsbewertung. Damit knüpft die Untersuchung unmittelbar an die Forschungsfrage **FF3.1** an, erweitert sie jedoch um eine zentrale Vergleichsdimension: Während die erste Untersuchung Rückmeldungen unter *Minimal-kontext* analysiert, untersucht die vorliegende Studie dieselben Phänomene unter *kontextreicher Promptgestaltung*. Erst durch diesen kontrollierten Kontextkontrast lässt sich die in **FF3.1** angelegte Frage nach der *Kontextabhängigkeit* von Charakteristika und Qualität LLM-generierter Rückmeldungen belastbar beantworten. Zugleich prüft die Studie mit Blick auf **FF3.2**, inwieweit sich Feedbacktypen unter kontextualisierten Bedingungen gezielt steuern lassen.

Im Unterschied zur ersten Untersuchung wurde das Kategoriensystem um zwei zusätzliche Kategorien ergänzt, um spezifische Effekte der Kontextualisierung methodisch erfassen zu können:

- `PERS` (Personalisierung): Erfasst, ob die Rückmeldung explizit auf Merkmale der konkreten Studierendenlösung eingeht – etwa durch die Verwendung im Code vorkommender Variablennamen oder durch die Bezugnahme auf spezifische Strukturmerkmale der individuellen Lösung.
- `COMPL`[1] (Übereinstimmung mit Aufgabenbeschreibung): Prüft, ob die Rückmeldung inhaltlich mit den Anforderungen und Einschränkungen der Aufgabenstellung übereinstimmt.

Die Kategorie `PERS` wird ergänzt, um zu erfassen, inwieweit Rückmeldungen explizit auf Merkmale der individuellen Studierendenlösung eingehen oder ob sie vorwiegend allgemeine Aspekte der Aufgabe bzw. der Musterlösung aufgreifen. Diese Unterscheidung ist vor dem Hintergrund der erweiterten Kontextbereitstellung relevant, da im Prompt neben der Studierendenlösung auch eine Musterlösung, ein Klassenskelett und ggf. Testfälle enthalten sind. Analysiert werden kann somit, ob die erzeugten Rückmeldungen spezifisch auf die Besonderheiten der jeweiligen Lösung Bezug nehmen (personalisierte Rückmeldung) oder eher generisch aufgabenspezifische Elemente reproduzieren. Die Kategorie `COMPL` wird eingeführt, um systematisch zu prüfen, ob die Rückmeldungen mit den in der Aufgabenstellung vorgegebenen Anforderungen und Restriktionen übereinstimmen. In der ersten Untersuchung zeigte sich, dass ein Teil der irreführenden Informationen (`MIS`) möglicherweise darauf zurückzuführen war, dass im Minimalprompt keine Aufgabenbeschreibung enthalten war und entsprechende Vorgaben (z. B. verbotene Bibliotheken oder konkrete Rückgabetypen) daher nicht berücksichtigt wurden. Durch die explizite Einbettung der Aufgabenbeschreibung in der zweiten Studie kann nun untersucht werden, ob und in welchem Umfang die erzeugten Rückmeldungen konsistent mit den spezifizierten Vorgaben ausfallen. Beide Kategorien tragen dem Umstand Rechnung, dass durch die Kontextualisierung des Prompts zusätzliche Bezugnahmen auf Aufgabe und Lösung möglich werden, die in der ersten Studie methodisch nicht erfassbar waren.

Zur systematischen Bearbeitung der Forschungsfrage **FF3.2** wurde eine zusätzliche Analyseebene eingeführt, in der geprüft wird, inwiefern die vom Modell erzeugten Rückmeldungen tatsächlich dem im Prompt spezifizierten Feedbacktyp entsprechen. Die Klassifikation erfolgte auf Grundlage der Feedbacktypologie nach Keuning, Jeuring und Heeren (2019), die bereits in Kapitel 4 in einer überarbeiteten Fassung Anwendung fand. Zwei unabhängige Personen analysieren die Rückmeldungen dahingehend, welche Feedbacktypen in ihnen enthalten waren, und ordnen diese mithilfe des Kategoriensystems zu. Uneinigkeiten wurden nach dem Prinzip der Konsensbildung aufgelöst (Hill, Thompson und Williams 1997; Hill et al. 2005).

Um eine unbewusste Verzerrung zu vermeiden, erfolgt die Kodierung zunächst *blind*. Das heißt, die kodierenden Personen arbeiten zunächst ohne Kenntnis des jeweils im Prompt spezifizierten Typs und analysieren die Rückmeldung ausschließlich auf Basis des inhaltlichen Gehalts. Erst

[1] Originalbezeichnung: *Compliance with Task Description*

in einem zweiten Schritt wird die Zuordnung mit dem intendierten Feedbacktyp abgeglichen. Auf diese Weise soll methodisch abgesichert werden, dass die Bewertung der Steuerbarkeit nicht durch Erwartungseffekte beeinflusst wird.

Mit dieser zusätzlichen Analyseebene geht die zweite Untersuchung über die erste hinaus: Während in ersterer lediglich spontane Vorkommen bestimmter Feedbacktypen erfasst wurden, wird hier explizit überprüft, ob und wie zuverlässig ein Sprachmodell die intendierte Typenvorgabe umsetzen kann.

6.3.2 Ergebnisse

Im Folgenden werden die Ergebnisse der zweiten Untersuchung dargestellt. Zunächst wird die Entwicklung des kontextualisierten Prompts mit zentralen Herausforderungen und Anpassungsschritten skizziert. Anschließend folgt die qualitative Analyse der generierten Rückmeldungen mit Fokus auf die Steuerbarkeit der Feedbacktypen.

Die Darstellung der Promptentwicklung ist Teil der Ergebnisschilderung, da der Prompt das zentrale Erhebungsinstrument darstellt. In der Logik der strukturierenden Inhaltsanalyse nach Mayring (2015a) gehört die *Dokumentation der Materialgenerierung* zur Ergebnistransparenz – eine genaue Beschreibung von Herkunft und Entstehungsbedingungen des Materials ist erforderlich, um die Analyse nachvollziehbar zu machen (Mayring 2014). Da der Prompt die Art und Qualität der Rückmeldungen maßgeblich prägt, werden die wesentlichen Iterationen sowie die finale Fassung als Ergebnisse ausgewiesen.

6.3.2.1 Kontextualisierter Prompt

Insgesamt waren fünf Iterationen erforderlich, bis ein Prompt vorlag, mit dem qualitativ vielversprechende Rückmeldungen generiert werden konnten, die als Grundlage für die anschließende Analyse dienten. Die folgenden Abschnitte skizzieren typische Probleme, Anpassungsschritte und erreichte Verbesserungen im Verlauf der Entwicklung. Abschließend wird die finale Version des Prompts vorgestellt, die im Rahmen der Analyse des erweiterten Datensatzes zum Einsatz kam.

Erste Iteration

Auf Grundlage der in Abschnitt 6.3.1.1 definierten Leitlinien sowie etablierten Prinzipien der Promptgestaltung (Zamfirescu-Pereira et al. 2023; OpenAI 2024) wurde zunächst eine initiale Version des Prompts entworfen. Dieser umfasste folgende Elemente:

- einen Systemprompt mit Hinweisen zur Zielgruppe, zur Einordnung der bereitgestellten Informationen sowie kurze Erläuterungen der verschiedenen Feedbacktypen nach Keuning, Jeuring und Heeren (2019),
- die Aufgabenbeschreibung (übersetzt ins Englische),
- das in der Aufgabenstellung enthaltene Klassengerüst,
- den Quellcode der Studierendeneinreichung,
- eine Musterlösung zur Aufgabe sowie
- eine Instruktion zur Erzeugung des jeweils geforderten Feedbacktyps (`KR`, `KP`, `KC`, `KTC`, `KM` oder `KH`).

Bei der Analyse der generierten Rückmeldungen zeigten sich drei zentrale Probleme: (P1) Rückmeldungen des Typs `KR` (Knowledge of Result) fielen mit einer Länge von durchschnittlich 290

Wörtern deutlich zu umfangreich aus, obwohl lediglich eine knappe Richtig/Falsch-Aussage mit maximal fünf Wörtern zu erwarten war (z. B. "Die Lösung ist falsch"). (P2) Viele Rückmeldungen enthielten neben dem angeforderten Feedback-Typ zusätzliche Elemente weiterer Typen, was die Typenklarheit erheblich einschränkte. (P3) Besonders kritisch war, dass zahlreiche Rückmeldungen explizit auf die mitgelieferte Musterlösung Bezug nahmen, etwa durch direkte Vergleiche mit der Studierendenlösung – ein Umstand, der bereits in früheren Studien als problematisch eingestuft wurde (Roest, Keuning und Jeuring 2024; Hellas et al. 2023).

`Veränderungen am Prompt:` Es wurden drei gezielte Modifikationen vorgenommen: Um (P1) zu begegnen, wurde für den Feedbacktyp `KR` eine klare Anweisung ergänzt, ausschließlich eine kurze Aussage zur Korrektheit der Lösung zu generieren (siehe Zeile 17 in Listing 6.1). Bzgl. (P2) wurde festgelegt, dass nur der explizit angeforderte Feedback-Typ erzeugt werden darf (Zeile 24). Für (P3) wurde eine Restriktion hinzugefügt, dass in der generierten Rückmeldung keinerlei inhaltliche Bezüge zur Musterlösung hergestellt werden darf (Zeile 25).

Zweite Iteration

In der zweiten Iteration zeigten sich bereits deutliche Verbesserungen gegenüber der ersten Generierung: Die Rückmeldungen fielen insgesamt kürzer aus, sodass sich `KR`-Feedback nun auf knappe Richtig/Falsch-Aussagen beschränkte (vgl. (P1)). Zudem war eine stärkere Orientierung am jeweils angeforderten Feedbacktyp erkennbar (vgl. (P2)). Problematische Verweise auf die Musterlösung (vgl. (P3)) traten in den neu erzeugten Ausgaben nicht mehr auf.

Gleichzeitig zeigten sich neue Probleme, die es zu adressieren galt: (P4) Rückmeldungen des Typs `KTC` (Knowledge about Task Constraints) bestanden teilweise lediglich in der wörtlichen Wiederholung von Elementen der Aufgabenbeschreibung – ohne einen erkennbaren Bezug zur konkreten Lösung der Lernenden. (P5) Rückmeldungen des Typs `KH` (Knowledge on How to Proceed) blieben stellenweise sehr vage und beschränkten sich auf allgemeine Aussagen wie "Überprüfe, ob das korrekt ist" – ohne elaborierte Hinweise oder konkrete nächste Schritte zu benennen. (P6) Beim Typ `KC` (Knowledge about Concepts) wurden vereinzelt Programmierkonzepte erläutert, die in keinem erkennbaren Zusammenhang mit dem tatsächlichen Fehler im Code standen. (P7) Rückmeldungen des Typs `KP` (Knowledge about Performance) erwiesen sich teils als sachlich unzutreffend, unklar formuliert oder reduzierten sich auf einfaches `KR`-Feedback.

`Veränderungen am Prompt:` Für (P4) wurde präzisiert, dass Rückmeldungen des Typs `KTC` nur dann auf die Aufgabenbeschreibung verweisen dürfen, wenn tatsächlich ein damit verbundenes Problem in der Studierendenlösung erkennbar ist (Zeile 21). Im Hinblick auf (P5) wurde ergänzt, dass `KH`-Rückmeldungen über allgemeine Prüfhinweise hinausgehen und konkrete Vorschläge für das weitere Vorgehen enthalten sollen (Zeile 20). Zur Lösung von (P6) wurde festgelegt, dass Konzepte nur dann erläutert werden dürfen, wenn sie in direktem Zusammenhang mit einem Problem in der Studierendenlösung stehen (Zeile 18). Schließlich wurde zur Adressierung von (P7) die Anweisung ergänzt, die Aufgabe in sinnvolle Teilziele zu untergliedern und den aktuellen Bearbeitungsstand prozentual zu bewerten (Zeile 19).

Dritte Iteration

Die dritte Iteration brachte teilweise qualitative Rückschritte mit sich: (P8) Rückmeldungen des Typs `KM` (Knowledge about Mistakes) enthielten zunehmend Korrekturhinweise und damit Elemente des Typs `KH`, obwohl `KM`-Feedback ausschließlich auf die Beschreibung eines Fehlers abzielen soll. (P9) Die generierten Rückmeldungen wurden wieder umfangreicher und wiesen eine zunehmend ausschweifende Sprache auf, was die Verständlichkeit insbesondere für Programmieranfängerinnen und -anfänger beeinträchtigen könnte. (P10) Motivationale Elemente (`MOT`) sowie (P11) erläuternde Beispiele (`EXA`) waren nur sehr selten vorhanden. Positiv hervorzuheben ist

hingegen, dass KP-Rückmeldungen nun nicht nur eine Prozentangabe zum Fortschritt enthielten, sondern zusätzlich eine erläuternde Begründung. Diese Ergänzung wurde als hilfreich eingestuft und soll in den weiteren Iterationen beibehalten werden, da sie zur Nachvollziehbarkeit der Einschätzung beitragen kann.

`Veränderungen am Prompt:` Zur Trennung von KM und KH (P8) wurde eine zusätzliche Instruktion eingeführt, dass KM-Rückmeldungen ausschließlich die Fehlerbeschreibung enthalten dürfen, während Hinweise zur Behebung dem Typ KH vorbehalten bleiben (Zeile 22). Im Hinblick auf (P9) wurde die Anweisung verschärft, Rückmeldungen sprachlich prägnant und adressatengerecht zu formulieren (Zeile 16). Darüber hinaus wurde ergänzt, dass Rückmeldungen der Typen KM, KTC, KC und KH nach Möglichkeit durch erläuternde Beispiele (EXA) unterstützt werden sollen (Zeile 23), um das Verständnis insbesondere bei abstrakteren Inhalten zu fördern. Diese Erweiterung ist als methodische Variation zu verstehen, mit der geprüft werden sollte, inwiefern sich durch gezielte Instruktionen eine höhere Frequenz und Qualität erläuternder Beispiele erreichen lässt.

Vierte Iteration

Die vierte Iteration zeigte, dass die zuvor eingeführten Änderungen nur begrenzte Verbesserungen im Output bewirkten. Rückmeldungen blieben teilweise sehr umfangreich und enthielten ausschweifende Erläuterungen (P9).

Bezüglich (P11) traten nun häufiger „Beispiele" auf, die jedoch inhaltlich sehr unterschiedlich ausgeprägt waren. Teilweise wurden lediglich Ausschnitte aus dem fehlerhaften Code der Studierenden nahezu unverändert wiedergegeben und als "Beispiel für einen Fehler" deklariert. Diese Form verfehlt jedoch den intendierten Zweck von KC-Feedback, das auf erklärende, verständnisfördernde Beispiele abzielt. Daneben fanden sich jedoch auch Fälle, in denen tatsächlich illustrative und kontextuell passende Beispiele erzeugt wurden, etwa: "Zum Beispiel würde Ihre aktuelle Implementierung bei `a = 1`, `b = 2`, `c = 3` ein SCALENE-Dreieck klassifizieren, obwohl diese Seiten die Dreiecksungleichung verletzen und daher kein gültiges Dreieck bilden."

`Veränderungen am Prompt:` In Reaktion auf diese Beobachtungen wurden zwei Anpassungen vorgenommen. Erstens wurde die Anweisung, auch für KM-Rückmeldungen Beispiele zu generieren, wieder entfernt, da die erzeugten Beispiele dort häufig nicht den intendierten Zweck erfüllten. Zweitens wurde die Instruktion zur Trennung von Fehlerbeschreibung (KM) und Korrekturhinweisen (KH) nochmals präzisiert, um klarzustellen, dass KM-Rückmeldungen ausschließlich die Problembeschreibung enthalten und keinerlei Hinweise zur Behebung geben dürfen (Zeile 22). Insgesamt verdeutlichte diese Iteration, dass die Einbindung von Beispielen ein differenzierteres Promptdesign erfordert, das im Rahmen dieser Untersuchung nicht vollständig adressiert werden konnte.

Fünfte Iteration und finaler Prompt

Die Ergebnisse der fünften Iteration zeigten, dass die in der vorangegangenen Runde identifizierten Probleme weitgehend behoben waren. Es traten keine neuen Auffälligkeiten auf, und die Qualität der Rückmeldungen entsprach im Wesentlichen dem Niveau der dritten Iteration, ohne dass erneut gravierende Schwächen beobachtet wurden. Da nicht davon auszugehen war, dass durch weitere Feinanpassungen substanzielle Verbesserungen erzielt werden könnten, wurde die in dieser Iteration verwendete Promptfassung als hinreichend stabil eingestuft und im Konsens als final festgelegt. Diese Version bildete anschließend die Grundlage für die Generierung der Rückmeldungen im erweiterten Datensatz. Der vollständige Prompt ist in Listing 6.1 dokumentiert.

```
1  Your task is to give feedback to a novice programmer who is
       working on a specific programming exercise in Java. You
       will be given (A) the complete task description as a
       LaTex Source Code, (B) the class body that is provided
       to the student, (C) the submitted solution of the
       student, (D) a model solution and (E) a desired feedback
        type with regard to the feedback typology described
       below.
2  Each feedback type is described with the name of the
       feedback type in wrapped in "" followed by a description
        of the feedback type wrapped in ~~
3
4  "Knowledge about task constraints (KTC)" ~ KTC are
       elaborated components that provide information of task
       rules, task constraints, and task requirements ~
5
6  "Knowledge about concepts (KC)" ~ KC are elaborated
       components that provide information on conceptual
       knowledge relevant for task processing ~
7
8  "Knowledge about mistakes (KM)" ~ KM are elaborated
       components that provide information on errors or
       mistakes ~
9
10 "Knowledge on how to proceed (KH)" ~ KH are elaborated
       components that provide information on procedural
       knowledge relevant for task processing ~
11
12 "Knowledge about performance (KP)" ~ KP provides summative
       feedback on the achieved performance level after doing
       multiple tasks or subtasks, such as '15 of 20 correct'
       and '85% correct' ~
13
14 "Knowledge of result/response (KR)" ~ KR is feedback that
       communicates whether a solution is correct or incorrect
       e.g. (1) programming solution passes all tests and
       contains no mistakes (2) programming solution is
       semantically equivalent to the model solution  ~
15
16 Try to make sure that the feedback you generate satisfies
       the following criteria:
17 - for feedback type KR the feedback should only contain the
       information if the student submission is correct or
       incorrect. Do not explain concepts or give information
       about errors or how to proceed.
18 - for feedback type KC only focus on programming concepts
       and explain these concepts only if the issue in the
       submitted solution of the student is related to that
       concept.
19 - for feedback type KP try to divide the task into useful
       subgoals and give feedback about the current state of
       the student submission in percentage.
20 - for feedback type KH not only tell the student to check if
        something is correct but provide hints and suggestions
       on how to proceed.
21 - for feedback type KTC only point to information in the
```

```
        task description if there is an issue in the student
        submission related to that task constraint.
22    - for KM Feedback just give the student information about
        the error and do not describe how to fix the error.
23    - make sure that the generated feedback corresponds exactly
        to the description of the desired feedback type. Do not
        provide any additional information that does not match
        the feedback type.
24    - the feedback should never contain information about the
        model solution. The student does not know the model
        solution.
25    - your feedback has to be brief and precise.
26
27    (A) complete task description:
28
29    ## INSERT Add task description here ##
30
31    (B) class body in file "##INPUT## Name of File" provided to
        the student:
32
33    ## INSERT Add Class Body, if provided ##
34
35    (C) student's submitted solution:
36
37    ## INSERT Add Student Solution ##
38
39    (D) model solution:
40
41    ## INSERT Model solution ##
42
43    (E) desired feedback type:
44
45    ## INSERT desired feedback type ##
```

Listing 6.1: Finaler Prompt zur Generierung spezifischer Feedback-Typen.

6.3.2.2 Generierung spezifischer Feedbacktypen

Die Ergebnisse der Analyse zur Generierung spezifischer Feedbacktypen sind in Spalte `FB TYPES` in Tabelle 6.6 dargestellt. Insgesamt wurden 66 Rückmeldungen kodiert und hinsichtlich ihrer tatsächlich realisierten Feedbacktypen (*Actual Feedback Type*, AFT) ausgewertet. Diese wurden anschließend mit den im Prompt spezifizierten Soll-Typen (*Desired Feedback Type*, DFT) verglichen.

In **63 von 66 Fällen** entsprach der generierte Inhalt dem angeforderten Feedbacktyp. Dies zeigt, dass das Modell (`GPT-4`) mit hoher Zuverlässigkeit in der Lage ist, die im Prompt spezifizierte Typanforderung umzusetzen. Gleichwohl enthielten **19 Rückmeldungen** zusätzlich Elemente weiterer Feedbacktypen. Besonders häufig betraf dies die Feedback-Typen `KH` (Knowledge on How to Proceed) und `KTC` (Knowledge about Task Constraints), die mit Komponenten von `KM` (Knowledge about Mistakes) kombiniert auftraten.

Eine Erklärung hierfür liegt in der inhaltlichen Verschränkung dieser Typen: Hinweise zum weiteren Vorgehen setzen häufig voraus, dass zuvor auf die zugrunde liegende Problemursache eingegangen wird. Ähnlich verhält es sich bei `KTC`-Feedback, das in vielen Fällen nur über die Beschreibung des fehlerhaften Verhaltens konkretisiert werden kann. Dass Feedbackformen nicht immer

trennscharf voneinander abzugrenzen sind, ist auch empirisch belegt: In spontanen Kommunikationssituationen treten unterschiedliche Feedbackarten häufig gemeinsam auf, die Übergänge sind fließend und wechselseitige Bezüge dominieren (Ditton 2014). Darüber hinaus könnten die beobachteten Typkombinationen auf typische Muster in den Trainingsdaten zurückgehen, da in Hilfeforen oder Lernressourcen häufig ein kombiniertes Vorgehen gewählt wird, bei dem zunächst ein Fehler identifiziert (`KM`) und anschließend ein Lösungsweg aufgezeigt wird (`KH`).

Insgesamt verdeutlicht die Analyse, dass das Modell in **95 % der Fälle** den angeforderten Feedbacktyp korrekt realisierte. Auch wenn sich die Trennschärfe zwischen bestimmten Typen – insbesondere zwischen `KM` und `KH` – nicht vollständig erzwingen ließ, belegen die Ergebnisse eine grundsätzlich hohe Steuerbarkeit der Typgenerierung durch kontextualisierte Prompts. Damit kann die Forschungsfrage **FF3.2** nach der gezielten Steuerbarkeit von Feedbacktypen positiv beantwortet werden: LLMs lassen sich durch explizite Promptgestaltung so anleiten, dass sie mit hoher Zuverlässigkeit differenzierte und typenspezifische Rückmeldungen erzeugen.

6.3.2.3 Charakteristika und Qualität der Rückmeldungen

Neben der Analyse der Steuerbarkeit wurde das erhobene Material auch inhaltlich ausgewertet. Ziel war es, typische Muster und Qualitätsmerkmale der generierten Rückmeldungen zu erfassen und systematisch zu beschreiben. Die Auswertung orientierte sich an den in Abschnitt 6.2.1.2 eingeführten Kategorien und umfasste sowohl inhaltliche Dimensionen (z. B. Stilhinweise, Beispiele, Codebezüge) als auch qualitative Aspekte wie sachliche Korrektheit oder das Auftreten irreführender Elemente. Darüber hinaus wurden metakognitive und motivationale Anteile sowie der Bezug zur individuellen Studierendenlösung (`PERS`) und zur Aufgabenbeschreibung (`COMPL`) kodiert.

Im Folgenden werden die Ergebnisse dieser Analyse in vier Teilbereichen dargestellt: (1) inhaltliche Muster, (2) Qualitätsmerkmale, (3) metakognitive und motivationale Elemente sowie (4) Personalisierung und Aufgabenbezug. Darauf aufbauend werden anschließend typenbezogene Befunde sowie querschnittliche Beobachtungen zu korrekten Lösungen, logischen Strukturen und sprachlich-stilistischer Ausgestaltung beschrieben.

(1) Inhaltliche Muster (`CONTENT`): In den Rückmeldungen zeigten sich nur vereinzelt **stilistische Hinweise** (`STYLE`): Insgesamt wurden sie in 9 von 66 Fällen kodiert und traten fast ausschließlich bei korrekten Lösungen auf. Nur in zwei Fällen bezogen sich Rückmeldungen mit Stil-Hinweisen auf fehlerhafte Einreichungen. Auffällig war zudem, dass solche Hinweise vor allem in Rückmeldungen für `KM`- und `KH`-Feedback vorkamen. Veranschaulichungen durch **Beispiele** (`EXA`) spielten dagegen kaum eine Rolle und ließen sich lediglich in einem einzigen Fall kodieren (vgl. Abschnitt zu `KC`). **Informationsanfragen** nach weiterem Kontext (`INFO`) traten in keiner der generierten Rückmeldungen auf. Bezüge auf Code waren zwar vorhanden, beschränkten sich jedoch meist auf kurze *Snippets*, die in 14 von 66 Rückmeldungen auftauchten. Vollständige Lösungsvorschläge wurden in keinem Fall kodiert.

(2) Qualitätsbezogene Muster (`QUALITY`): Die Ergebnisse der Analyse zeigen, dass **irreführende Inhalte** auch bei erweitertem Kontext weiterhin auftreten. Insgesamt enthielten 18 % der generierten Rückmeldungen Fälle mit sachlich unzutreffenden Aussagen. So wurde etwa in einer Rückmeldung zu einer Lösung mit korrekter `if-else`-Struktur fälschlicherweise eine "inkonsistente Verwendung von Klammern" bemängelt, obwohl im Studierenden-Code keinerlei Unstimmigkeiten vorlagen. Weitere Beispiele für fehlerhafte oder missverständliche Rückmeldungen finden sich in den nachfolgenden Abschnitten zu den einzelnen Feedbacktypen, wo sie jeweils im Detail beschrieben und eingeordnet werden. Auf die Aufgabenebene bezogen traten irreführende Inhalte in 5 von 11 Aufgaben auf. **Unsicherheitsmarkierungen** (`UNC`) wurden in keiner Rückmeldung kodiert.

Tabelle 6.6: Ergebnisse der Kodierung der generierten Rückmeldungen mit erweitertem Kontext

	FB TYPES		CONTENT						QUALITY			OTHER			
ID	**DFT**	**AFT**	**INFO**	**STYLE**	**CAUSE**	**FIX**	**CODE**	**EXA**	**COMP**	**MIS**	**UNC**	**META**	**MOT**	**PERS**	**COMPL**
10_TEOS	KR	KR	○	○	○	○	○	○	—	○	○	○	○	●	○
	KTC	KTC	○	○	●	●	○	○	—	○	○	○	○	●	●
	KC	KC, KM	○	○	●	●	○	○	—	○	○	○	○	●	●
	KM	KM	○	○	●	●	○	○	—	○	○	○	○	●	●
	KH	KH	○	○	●	●	○	○	—	○	○	○	○	●	●
	KP	KP	○	○	●	○	○	○	—	○	○	○	○	●	●
13_NEGF	KR	KR	○	○	○	○	○	○	—	○	○	○	○	●	○
	KTC	KTC, KM	○	○	●	●	○	○	—	○	○	○	○	●	●
	KC	KC, KM	○	○	●	●	◐	○	—	○	○	○	○	●	●
	KM	KM	○	○	●	●	◐	○	—	○	○	○	○	●	●
	KH	KH, KM	○	○	●	●	◐	○	—	○	○	○	○	●	●
	KP	KP, KM	○	○	●	●	○	○	—	○	○	○	○	●	●
01_TTBSA	KR	KR	○	○	○	○	○	○	—	○	○	○	○	●	○
	KTC	KTC	○	○	●	●	○	○	—	○	○	○	○	●	●
	KC	KC, KM	○	○	●	●	○	○	—	○	○	○	○	●	●
	KM	KM	○	○	●	●	○	○	—	○	○	○	○	●	●
	KH	KH	○	○	●	●	○	○	—	○	○	○	○	●	●
	KP	KP, KM	○	○	●	○	○	○	—	○	○	○	○	●	●
FIB_01	KR	KR	○	○	○	○	○	○	—	○	○	○	○	●	○
	KTC	KTC	○	○	●	○	◐	○	—	○	○	○	○	●	●
	KC	KC, KM	○	○	●	○	○	○	—	○	○	○	○	●	●
	KM	KM, KH	○	○	●	●	◐	○	—	○	○	○	○	●	●
	KH	KH, KM	○	○	●	●	◐	○	—	○	○	○	○	●	●
	KP	KP, KM	○	○	●	○	◐	○	—	○	○	○	●	●	●
FIB_02 ✓	KR	KR	○	○	○	○	○	○	—	○	○	○	○	●	○
	KTC	KTC	○	○	○	○	○	○	—	○	○	○	○	○	●
	KC	KC	○	○	○	○	○	○	—	○	○	○	○	●	●
	KM	KM, KH	○	○	●	●	◐	○	—	●	○	○	○	●	●
	KH	KH, KC	○	○	○	○	○	○	—	○	○	○	○	○	○
	KP	KP, KM	○	○	○	○	○	○	—	○	○	○	○	●	●
PASS_01	KR	KR	○	○	○	○	○	○	—	○	○	○	○	●	○
	KTC	KTC, KM, KH	○	○	●	●	○	○	—	○	○	○	○	●	●
	KC	KC, KM	○	○	●	○	◐	○	—	○	○	○	○	●	●
	KM	KM	○	○	●	○	◐	○	—	○	○	○	○	●	●
	KH	KH	○	○	○	●	○	○	—	○	○	○	○	●	●
	KP	KP, KM	○	○	●	○	○	○	—	○	○	○	○	●	●
MAX3_01	KR	KR	○	○	○	○	○	○	—	●	○	○	○	●	○
	KTC	KTC	○	○	○	○	○	○	—	●	○	○	○	●	●
	KC	KC, KM	○	○	●	●	○	○	—	○	○	○	○	●	●
	KM	KM	○	○	●	○	◐	●	—	●	○	○	○	●	●
	KH	KH	○	●	○	●	○	○	—	○	○	○	○	●	●
	KP	KP, KM	○	○	○	●	○	○	—	●	○	○	○	●	●
MAX3_02 ✓	KR	KR	○	○	○	○	○	○	—	○	○	○	○	●	○
	KTC	KTC	○	○	●	●	○	○	—	●	○	○	○	●	○
	KC	KC	○	●	○	○	◐	○	—	○	○	○	○	●	●
	KM	KM, KH	○	●	●	●	◐	○	—	●	○	○	○	●	●
	KH	KH	○	●	○	●	○	○	—	●	○	○	○	●	●
	KP	KP, KM	○	○	○	○	○	○	—	●	○	○	○	●	●
MAX3_03 ✓	KR	KR	○	○	○	○	○	○	—	○	○	○	○	●	○
	KTC	KTC	○	○	○	○	○	○	—	●	○	○	○	●	●
	KC	KC	○	○	○	○	○	○	—	○	○	○	○	●	●
	KM	KM, KH	○	●	●	●	○	○	—	○	○	○	○	●	●
	KH	KH	○	●	●	●	○	○	—	●	○	○	○	●	●
	KP	KP, KM	○	○	○	○	○	○	—	●	○	○	○	●	●
MAX3_04 ✓	KR	KR	○	○	○	○	○	○	—	○	○	○	○	●	○
	KTC	KTC	○	○	●	○	○	○	—	●	○	○	○	●	●
	KC	KC	○	○	○	○	○	○	—	○	○	○	○	●	●
	KM	KM	○	●	●	○	○	○	—	○	○	○	○	●	●
	KH	KH	○	●	○	●	○	○	—	○	○	○	○	●	●
	KP	KP	○	○	○	○	○	○	—	○	○	○	○	●	●
MAX3_05	KR	KR	○	○	○	○	○	○	—	○	○	○	○	●	○
	KTC	KM, KH	○	○	●	●	○	○	—	○	○	○	○	●	●
	KC	KC	○	●	○	○	◐	○	—	○	○	○	○	●	●
	KM	KM	○	○	●	○	○	○	—	○	○	○	○	●	●
	KH	KH, KM	○	○	●	●	○	○	—	○	○	●	○	●	●
	KP	KP, KM	○	○	○	○	○	○	—	○	○	○	○	●	●

●: Ja ○: Nein ◐: Snippet —: Nicht anwendbar ✓: Musterlösung **DFT**/**AFT**: erwarteter/tatsächlicher FB-Typ

(3) Metakognitive und motivationale Elemente (OTHER): Metakognitive Hinweise (META) fanden sich nur in einer Rückmeldung zur Aufgabe MAX3 bei angefordertem KH. **Motivationale Elemente** (MOT) traten ebenfalls nur in einer Rückmeldung auf.

(4) Personalisierung und Übereinstimmung mit Aufgabenbeschreibung (OTHER): Die Ergebnisse zeigen, dass in 64 von 66 der generierten Rückmeldungen die Kategorie PERS (Personalisierung) kodiert wurde. Dies bedeutet, dass die Rückmeldungen überwiegend spezifische Merkmale der individuellen Studierendenlösung aufgriffen, etwa durch die Verwendung konkreter Variablennamen oder durch Bezugnahme auf die Struktur der eingereichten Lösung.

In 53 von 66 Rückmeldungen wurde die Kategorie COMPL (Übereinstimmung mit Aufgabenbeschreibung) kodiert. Dies bedeutet, dass die Rückmeldungen inhaltlich mit den Anforderungen und Einschränkungen der Aufgabenstellung übereinstimmten. Dieses Ergebnis ist jedoch um die Fälle zu bereinigen, bei denen der Feedbacktyp KR angefordert wurde, da diese Rückmeldungen explizit nur richtig/falsch ausgeben sollten. Somit wurde in 54 von 55 Rückmeldungen, zu denen inhaltliche Rückmeldungen generiert wurden, die Übereinstimmung mit der Aufgabenbeschreibung kodiert. Dies zeigt, dass das Modell in der Lage ist, Rückmeldungen zu generieren, die den spezifischen Anforderungen der Aufgabenstellung entsprechen.

Typenspezifische Befunde und querschnittliche Beobachtungen

Über diese querschnittlichen Muster hinaus werden im Folgenden die Ergebnisse *typenbezogen* präzisiert. Dabei wird dargestellt, wo die Rückmeldungen konsistent Stärken zeigen (z. B. strukturierte Teilziel-Bewertungen bei KP) und wo weiterhin Unschärfen oder Fehlsteuerungen auftreten. Ergänzend werden querschnittliche Aspekte wie der Umgang mit korrekten Lösungen, logischen Strukturen sowie Sprache und Tonalität eingeordnet.

KR-Feedback

Bis auf einen Fall war die Rückmeldung zu richtig/falsch korrekt. In einem Fall wurde eine korrekte Lösung fälschlicherweise als inkorrekt bewertet. Die Rückmeldungen enthielten alle, wie in der Promptvorgabe gefordert, ausschließlich eine knappe Aussage zur Korrektheit der Lösung, ohne weiterführende Erläuterungen oder Hinweise. Die Länge der Rückmeldungen variierte zwischen 2 und 5 Wörtern, was den Vorgaben entsprach.

KTC-Feedback

In den Rückmeldungen zu Aufgaben mit expliziten Vorgaben fanden sich wiederholt zutreffende Hinweise auf konkrete Anforderungen der Aufgabenstellung (Subtyp: KTC-TR – *Hints on task requirements*). So wurde etwa bei der Aufgabe TEOS korrekt auf die Pflicht zur Nutzung der Konstantenklasse verwiesen:
"`The task specifically requires you to use the constants defined in the PhysicsConstants class for these values to ensure consistency and maintainability of the code.`"
Ein vergleichbares Muster zeigte sich bei Rückmeldungen zur Aufgabe NEGF, bei der die verpflichtende Verwendung der *generalized Fibonacci sequence formula* hervorgehoben wurde:
"`The task explicitly requires the use of the generalized Fibonacci sequence formula for negaFibonacci, without resorting to alternative formulas or approaches.`"

Daneben traten auch Rückmeldungen auf, die eher allgemeine Hinweise zum Vorgehen enthielten (Subtyp: KTC-TPR – *Hints on task-processing rules*). Häufig handelte es sich dabei jedoch lediglich um Umformulierungen der Aufgabenstellung, ohne dass ein erkennbarer didaktischer Mehrwert entstand.

Problematisch war zudem, dass bei Aufgaben ohne explizite Vorgaben – wie etwa MAX3 – vermeintliche KTC-Hinweise generiert wurden. So wurde bei korrekten Lösungen fälschlich bemängelt: "Your program should output only the largest number, without any preceding text". Da in der Aufgabenstellung keinerlei entsprechende Vorgaben existierten, wurden korrekte Lösungen hier irrtümlich problematisiert.

Schließlich zeigte sich, dass die Abgrenzung zwischen KTC- und KM-Feedback nicht immer eindeutig gelang. In einigen Rückmeldungen wurden Anforderungen thematisiert, die zwar implizit aus der Aufgabenstellung ableitbar waren, jedoch nicht explizit formuliert wurden. In diesen Fällen wurde eine Kodierung als KM vorgenommen, da der Schwerpunkt auf der Beschreibung eines konkreten Fehlers in der Lösung lag.

Insgesamt wird deutlich, dass das Modell prinzipiell in der Lage ist, KTC-Feedback zu erzeugen, zugleich aber zur Übergeneralisierung neigt und in Aufgaben ohne explizite Vorgaben unzutreffende Hinweise generieren kann.

KC-Feedback

Die Rückmeldungen des Typs *Knowledge about Concepts* (KC) konzentrierten sich überwiegend auf fachliche Programmierkonzepte und adressierten in vielen Fällen inhaltlich passende Aspekte. Bei fehlerhaften Lösungen wurde meist ein relevantes Konzept benannt, das in direktem Zusammenhang mit dem vorliegenden Problem stand, während bei korrekten Lösungen eher allgemeine Hinweise zu zentralen Konzepten gegeben wurden – teilweise mit Bezug zur konkreten Umsetzung im Programmcode. In einzelnen Fällen traten Überschneidungen zu anderen Feedbacktypen auf, insbesondere zu KH-Elementen, wenn zusätzlich Korrekturhinweise enthalten waren.

Ein Beispiel hierfür findet sich bei einer fehlerhaften Lösung zur Aufgabe NEGF, wo das KC-Feedback ausführlich das Prinzip der Rekursion erklärte:
" each function call should ideally break down the problem into smaller, more manageable subproblems that resemble the original problem".
Die Rückmeldung enthielt zugleich Komponenten von CAUSE und FIX, da der konkrete Fehler im Zusammenhang mit der Rekursionsanwendung benannt wurde.

Auch bei korrekten Lösungen wurden relevante Konzepte thematisiert. So wurde in einer Rückmeldung zu einer Kotlin-Lösung der Aufgabe MAX3 die Verwendung verschachtelter max-Funktionen hervorgehoben:
"You've nested two max functions to compare three values: max(max(main_v2, main_v3), main_v4). This is a valid approach to find the largest of three numbers [...]"
Außerdem wurden Hinweise auf alternative idiomatische Sprachkonstrukte, wie den Einsatz von if- und when-Ausdrücken ergänzt. Weitere genannte Konzepte betrafen unter anderem Header-Dateien in C++, Vergleichsoperatoren, Konstanten und den Geltungsbereich von Variablen. Dieses Muster deutet darauf hin, dass bei korrekten Lösungen bevorzugt kontextuell relevante Konzepte retrospektiv erläutert wurden, gewissermaßen als zusammenfassende Reflexion über zentrale Aspekte der Lösung.

Ein Untertyp des KC-Feedbacks betrifft die Veranschaulichung durch Beispiele (KC-EXA). Solche illustrierenden Beispiele, die das konzeptuelle Verständnis fördern sollen, blieben nahezu vollständig aus. Zwar fanden sich in mehreren Rückmeldungen kurze Code-*Snippets*, vor allem in KM-, KH- und KTC-Feedback, jedoch stammten diese Ausschnitte meist direkt aus der Studierendenlösung oder zeigten einfache Syntaxelemente – ohne zur Verdeutlichung abstrakter Konzepte beizutragen. Daher wurden sie nicht als illustrierende KC-Beispiele kodiert.

Zusammenfassend zeigt sich, dass KC-Feedback inhaltlich häufig passende Konzepte adressierte, dabei jedoch eher erklärend als veranschaulichend ausfiel. Die systematische Integration von Beispielen zur Unterstützung des konzeptuellen Verständnisses blieb weitgehend aus.

KM-Feedback

Die Rückmeldungen des Typs *Knowledge about Mistakes* (KM) enthielten fast durchgängig eine CAUSE-Komponente mit einer expliziten Beschreibung des vermuteten Fehlers sowie eine PERS-Komponente mit direktem Bezug zur Studierendenlösung. Die Beschreibungen fielen überwiegend ausführlich aus und lieferten detaillierte Hinweise auf die vermutete Problemstelle. Auffällig war, dass trotz der klaren Anweisung im Prompt, auf Korrekturhinweise zu verzichten, in mehreren Fällen dennoch FIX-Elemente enthalten waren. Gerade bei komplexeren Aufgaben wurde vereinzelt nicht nur der Fehler benannt, sondern zugleich eine mögliche Lösungsskizze angedeutet.

Inhaltlich erwies sich das KM-Feedback größtenteils als korrekt, einzelne Rückmeldungen enthielten jedoch sachlich unzutreffende oder missverständliche Aussagen. Ein Beispiel betrifft eine korrekte Java-Lösung zur Aufgabe FIB, in der fälschlich eine fehlerhafte Reihenfolge der Basisfallprüfungen bemängelt wurde: "Your solution correctly implements the Fibonacci sequence; however, there's a mistake in the order of your base case checks. You should check for `n == 0` before `n == 1`...". Die beanstandete Reihenfolge beeinflusst die funktionale Korrektheit des Programms jedoch nicht und stellt somit keinen tatsächlichen Fehler im Rahmen der Aufgabenstellung dar.

Darüber hinaus zeigte sich, dass der Begriff "Fehler" teilweise auch für Aspekte verwendet wurde, die die Programmlogik nicht beeinträchtigen. Besonders bei korrekten Lösungen wurden formale Eigenschaften wie Stilfragen oder alternative Sprachkonstrukte problematisiert und als Fehler deklariert. Dies ist insofern problematisch, als Lernende den Eindruck gewinnen könnten, tatsächlich einen inhaltlichen Fehler begangen zu haben, obwohl lediglich Verbesserungsvorschläge adressiert wurden. Vermutlich hängt dies mit der Vorgabe zusammen, in jedem Fall Fehler im Code zu identifizieren. In Situationen ohne funktionale Probleme werden dann offenbar stilistische Auffälligkeiten oder Antipatterns als Fehler markiert, obwohl sie aus didaktischer Perspektive eher Optimierungshinweise darstellen.

KH-Feedback

Die Rückmeldungen des Typs *Knowledge on How to Proceed* (KH) konnten durchgängig der Kategorie FIX zugeordnet werden, da sie Anleitungen zum weiteren Vorgehen enthielten. In vielen Fällen bestanden diese Hinweise aus hilfreichen, klar strukturierten Handlungsschritten, die Lernenden aufzeigten, wie erkannte Probleme im Code gezielt behoben werden können. Dieses Ergebnis erscheint folgerichtig, da die FIX-Kategorie unmittelbar aus dem KH-Feedbacktyp abgeleitet wurde (vgl. Abschnitt 6.2.1.2).

Daneben fanden sich jedoch zahlreiche Rückmeldungen, die nur sehr allgemein formulierte Hinweise gaben und ohne konkrete nächste Schritte blieben. Solche Aussagen entsprachen funktional eher einer "think harder"-Rückmeldung, etwa:
"`Ensure your method correctly implements the formula provided in the task description.`"
In diesen Fällen blieb unklar, wie Lernende ihr Vorgehen konkret anpassen können, um die gestellten Anforderungen erfolgreich umzusetzen.

Ein möglicher Grund für diese Oberflächlichkeit liegt in der Instruktion, ausschließlich KH-Feedback zu generieren und keine Informationen über den konkreten Fehler (KM) bereitzustellen. Gerade bei komplexeren Aufgaben oder wenig differenzierten Lösungen führte dies offenbar dazu, dass

die Rückmeldungen auf generische Formulierungen auswichen, anstatt differenzierte Handlungsanleitungen zu bieten.

Auffällig war zudem, dass KH-Feedback häufig mit KM-Elementen kombiniert wurde, obwohl der Prompt explizit vorgab, nur den gewünschten Feedbacktyp zu erzeugen. In mehreren Fällen wurde nicht nur erläutert, wie vorzugehen sei, sondern zugleich der zugrunde liegende Fehler beschrieben. Interessanterweise zeigte sich dabei ein gegenläufiges Muster: Während ein Teil der KH-Rückmeldungen sehr allgemein blieb, enthielten die differenziertesten, schrittweise aufgebauten Anleitungen fast immer auch Fehlerbeschreibungen. Dies weist darauf hin, dass präzise prozedurale Hilfestellungen schwer von inhaltlichen Fehlererklärungen zu trennen sind.

KP-Feedback

Rückmeldungen des Typs *Knowledge about Performance* (KP) wurden gezielt daraufhin untersucht, ob sie über eine einfache Bewertung hinausgehen und eine strukturierte Einschätzung auf Basis von Teilzielen enthalten. Im Prompt war explizit vorgegeben, die Bewertung nicht nur als Prozent- oder Punktzahl auszugeben, sondern diese anhand von *Teilzielen* zu begründen – ein Vorgehen, das in der Lehr-Lernforschung zur Programmierausbildung als besonders lernförderlich gilt (Margulieux, Catrambone und Guzdial 2016).

Die Ergebnisse zeigen, dass diese Vorgabe in den meisten Fällen erfolgreich umgesetzt wurde. So wurden insbesondere bei der Aufgabe MAX3 sinnvolle Teilziele identifiziert, etwa: "(1) korrekte Einlesung der Eingaben, (2) fehlerfreier Vergleich der Werte, (3) korrekte Ausgabe des Ergebnisses". Bei korrekten Lösungen erfolgte in der Regel eine Bewertung von 100 %, ergänzt durch eine nachvollziehbare Begründung, inwiefern die definierten Teilziele erfüllt wurden.

Allerdings traten auch kleinere Inkonsistenzen auf. Mitunter erhielten weniger relevante Teilziele ein überproportional hohes Gewicht, wodurch die Gesamtbewertung verzerrt wurde. Zudem wurde in einem Fall eine vollständig korrekte Lösung lediglich mit 80 % bewertet, ohne dass diese Bewertung sachlich nachvollziehbar begründet war. Dies deutet darauf hin, dass die numerische Bewertung des Modells nicht durchgängig konsistent erfolgt und vereinzelt zu Über- oder Unterbewertungen führen kann.

Rückmeldungen zu korrekten Lösungen

Im Gegensatz zur ersten Untersuchung wurden in dieser Studie auch Rückmeldungen zu korrektem Programmcode generiert. Ziel war es zu prüfen, inwieweit Rückmeldungen konsistent als korrekt ausgewiesen werden oder ob dennoch fehlerhafte Kommentare entstehen, wenn explizit Feedback zu (nicht vorhandenen) Fehlern angefordert wird.

Die Ergebnisse zeigen, dass dies nur eingeschränkt gelingt. Auch bei klar korrekten Lösungen traten mehrfach irreführende oder sachlich unzutreffende Rückmeldungen auf. Ein Beispiel betrifft eine korrekte Kotlin-Lösung zur Aufgabe MAX3 (vgl. Listing 6.2). Obwohl in der Lösung korrekt drei Ganzzahlen eingelesen werden und das Maximum berechnet, wurde im KTC-Feedback fälschlich die Art der Eingabe kritisiert: "... your current implementation reads three integers in a single line, which may not align with the test cases provided in the task description where inputs are expected line by line." Zudem wurde eine Überprüfung der Eingabemethode empfohlen, obwohl in der Aufgabenstellung keinerlei Vorgaben zu einer zeilenweisen Eingabe enthalten waren.

```
//..
import kotlin.math.max

fun main() {
    val main_v1 = Scanner(System.`in`)
    val main_v2 = main_v1.nextInt()
    val main_v3 = main_v1.nextInt()
    val main_v4 = main_v1.nextInt()
    println(max(max(main_v2, main_v3), main_v4))
}
```

Listing 6.2: Korrekte Kotlin-Einreichung zur Aufgabe MAX3.

Ähnliche Fehlklassifikationen traten bei korrekten Lösungen in `C++` und `Java` zur gleichen Aufgabe auf. Dort enthielt das `KTC`-Feedback beispielsweise den Hinweis, dass das Programm keinen erläuternden Text ausgeben, sondern nur den Zahlenwert darstellen würde – obwohl genau diese knappe Ausgabe der Aufgabenstellung entsprach.

Auch bei anderen Feedbacktypen wurden problematische Tendenzen deutlich. So enthielten Rückmeldungen zur korrekten `Kotlin`-Lösung zwar sinnvolle Stilhinweise (z. B. zur Verwendung von `readLine()` für idiomatischeren Code), klassifizierten diese im `KM`-Feedback jedoch explizit als "Fehler". Dies ist insofern irreführend, als es sich lediglich um stilistische Empfehlungen handelt und nicht um tatsächliche Programmfehler. Im `KP`-Feedback wurde der korrekten Lösung zudem nur ein Leistungsfortschritt von 85 % zugeschrieben, ohne nachvollziehbare inhaltliche Begründung.

Ein weiteres Beispiel betrifft eine korrekte `C++`-Lösung, bei der im `KC`-Feedback behauptet wurde, es werde der falsche Header für die Funktion `max()` verwendet, obwohl ein solcher Import weder notwendig noch fehlerhaft war. Das zugehörige `KM`-Feedback griff denselben Punkt auf, deklarierte ihn jedoch als stilistisches Problem mit dem Hinweis: "Removing unused libraries helps to keep the codebase more maintainable and potentially reduces the compilation time."

Positiv hervorzuheben ist, dass bei formal korrekten, aber unnötig komplexen Lösungen teilweise hilfreiche Optimierungsvorschläge generiert wurden. So enthielten Rückmeldungen zu einer korrekten, aber übermäßig verschachtelten `Java`-Lösung zur `MAX3`-Aufgabe Empfehlungen, die Bedingungsstruktur klarer zu gliedern: "Consider structuring your conditional statements to more clearly identify the largest number among the three inputs. Start by comparing the first number with the second and third numbers to determine if it's the largest. If it's not, move on to compare the second number with the third. This way, you can avoid redundant comparisons and make your logic more straightforward." Solche Rückmeldungen können dazu beitragen, Lernenden nicht nur Korrektheit, sondern auch Aspekte von Codequalität und Lesbarkeit näherzubringen – sofern sie als Stilhinweise und nicht als Fehlermeldungen verstanden werden.

Umgang mit logischen Strukturen

Ein bekanntes Defizit großer Sprachmodelle liegt in den begrenzten Fähigkeiten zur Analyse und Bewertung logischer Strukturen (Creswell, Shanahan und Higgins 2022). Dieses Muster zeigte sich auch in der vorliegenden Untersuchung, insbesondere im Umgang mit untypischen Lösungsstrategien von Studierenden.

Ein exemplarischer Fall betrifft eine fehlerhafte `Java`-Lösung zur Aufgabe `MAX3` (vgl. Listing 6.3), in der der Zahlenvergleich nicht mit den üblichen Vergleichsoperatoren (`>`, `<`) umgesetzt wurde, sondern durch Subtraktion und anschließende Prüfung des Vorzeichens. Dieses Vorgehen wich

deutlich von gängigen Lösungsmustern ab und ist im Kontext der Programmierausbildung ein eher ungewöhnlicher, aber grundsätzlich legitimer Ansatz.

```
//..
main_v1 = main_v7.nextInt();
main_v2 = main_v7.nextInt();
main_v3 = main_v7.nextInt();
main_v4 = main_v1 - main_v2;
if (main_v4 <=0){
    if ((main_v2 - main_v3)<=0){
        System.out.println(main_v3);
    } else {
        System.out.println(main_v2);
    }
}
```

Listing 6.3: Fehlerhafte Java-Einreichung zur Aufgabe MAX3 mit untypischer Logik.

Die zugehörigen Rückmeldungen machten deutlich, dass logische Fehler häufig nicht präzise identifiziert wurden. Stattdessen wurden generische Kritikpunkte formuliert, etwa der Hinweis, dass ein zusätzlicher Zweig zur Abdeckung eines speziellen Falls fehle: "your solution currently lacks a branch to handle the situation where the first number is less than the second but greater than the third". Ein solcher Fall war jedoch in der Aufgabenstellung gar nicht vorgesehen. Ergänzend wurde empfohlen, die Fallunterscheidung zu erweitern ("adding this condition will make your solution more comprehensive"), obwohl dies weder erforderlich noch sinnvoll gewesen wäre.

Diese Beobachtungen verdeutlichen, dass bei untypischen oder fehlerhaften logischen Konstruktionen oftmals auf allgemeine Plausibilitätsmuster zurückgegriffen wird, anstatt die tatsächliche Fehlerursache präzise zu benennen. Dadurch entstehen Rückmeldungen, die für Lernende wenig hilfreich sind oder sogar in die Irre führen können.

Sprache und Tonalität der Rückmeldungen

Die Rückmeldungen waren überwiegend in einer adressatengerechten, verständlichen Sprache formuliert und klar an *novice programmers* gerichtet, wie es im Prompt explizit angegeben war. In Einzelfällen wurden jedoch Fachbegriffe verwendet, die für Programmieranfängerinnen und -anfänger als anspruchsvoll einzustufen sind, etwa "variable shadowing" oder "data type mismatch". In diesen Fällen wurde der Begriff jedoch in der Regel durch eine anschließende Erklärung näher erläutert, sodass die Verständlichkeit insgesamt gewahrt blieb.

Schließlich ist hervorzuheben, dass keine herabwürdigenden oder unangemessenen Formulierungen beobachtet wurden. Die Rückmeldungen waren durchweg grammatikalisch korrekt und zeichneten sich durch einen neutralen bis freundlichen Ton aus, was ihre Eignung für die intendierte Zielgruppe zusätzlich unterstreicht.

6.4 Diskussion der Ergebnisse

Die vorliegenden Ergebnisse verdeutlichen, dass große Sprachmodelle grundsätzlich in der Lage sind, inhaltlich reichhaltige und formal strukturierte Rückmeldungen zu fehlerhaften Programmierlösungen zu generieren – selbst bei minimalem Kontext. Insbesondere die häufige Präsenz von Ursachenanalysen und Korrekturvorschlägen verweist auf ein beachtliches Potenzial zur Unterstützung formativer Rückmeldungsprozesse in der Programmierausbildung. In vielen Fällen

überstieg die inhaltliche Tiefe der generierten Rückmeldungen deutlich jene von etablierten automatisierten Systemen, die sich oftmals auf die Wiedergabe von Compilerfehlern oder Testergebnissen beschränken (Jeuring et al. 2022; Keuning, Jeuring und Heeren 2019).

Gleichzeitig weisen die Rückmeldungen auf deutliche Limitationen hinsichtlich der Qualität hin. In beiden Untersuchungen zeigte sich eine hohe inhaltliche Varianz der generierten Rückmeldungen, die teilweise gravierende sachliche Fehler, inkonsistente Hinweise und fehlende Bezugnahmen auf aufgabenspezifische Anforderungen enthielten. Solche Unschärfen sind aus didaktischer Sicht problematisch, da insbesondere Lernende im Anfangsunterricht häufig nicht über die metakognitiven Kompetenzen verfügen, um bereitgestelltes Feedback kritisch zu prüfen oder zu hinterfragen (Kiesler 2020b; Anderson und Krathwohl 2001).

Auffällig ist darüber hinaus die geringe Präsenz metakognitiver und motivierender Elemente. Die generierten Rückmeldungen konzentrierten sich überwiegend auf Fehlerkorrekturen und konkrete Problemlösungen, ohne Anregungen zur Reflexion oder Strategienentwicklungen zu bieten. Damit bleiben jene Aspekte unberücksichtigt, die in der empirischen Untersuchung in Kapitel 4 als charakteristisch für professionelles Lehrendenfeedback identifiziert wurden und nachweislich die Selbstregulation fördern und wesentlich zu nachhaltigen Lernprozessen beitragen (Narciss 2007). Die vorliegenden Ergebnisse zeigen somit, dass derartige qualitativ anspruchsvolle Komponenten ohne gezielte Instruktion im Prompt kaum spontan generiert werden. Für den didaktisch fundierten Einsatz großer Sprachmodelle bedeutet dies, dass metakognitive und motivationale Dimensionen entweder explizit durch die Promptgestaltung angeregt oder über ergänzende Systemmechanismen integriert werden müssen, um dem Niveau menschlicher Expertenrückmeldungen näherzukommen.

Ein weiterer Befund betrifft die geringe Reproduzierbarkeit der Modellantworten. Selbst bei identischer Eingabe zeigten sich deutliche Varianzen in Struktur und Inhalt der Rückmeldungen. Diese zufallsbedingte Streuung erschwert die didaktisch verlässliche Einbindung solcher Systeme in geplante Feedbackprozesse, insbesondere wenn kein erweiterter Kontext bereitgestellt wird. Zudem traten problematische oder sachlich falsche Rückmeldungen vor allem bei komplexeren Aufgaben (z. B. NEGF) gehäuft auf, was auf eine hohe Spezifität der Modellleistung in Bezug auf die Art der Lernaufgabe hinweist. Ähnliche Beobachtungen wurden auch in anderen Studien berichtet, die auf irreführende oder unvollständige Feedbacknachrichten verweisen (Azaiz, Kiesler und Strickroth 2024; Azaiz, Deckarm und Strickroth 2023; Kiesler, Lohr und Keuning 2023; Roest, Keuning und Jeuring 2024).

In Bezug auf **FF3.1** zeigen die Ergebnisse eine deutliche Qualitätssteigerung, wenn Kontextanreicherung erfolgt: Die Anzahl sachlich falscher oder irreführender Rückmeldungen nahm gegenüber der ersten Studie deutlich ab. Im kontextarmen Setting lag der Anteil solcher irreführender Aussagen bei über 60 % (vgl. Abschnitt 6.2), während sich diese Quote unter kontextualisierten Bedingungen spürbar verringerte (18 %). Ähnliche Beobachtungen zur Fehleranfälligkeit bei unstrukturiertem Prompting wurden auch in früheren Arbeiten berichtet (Azaiz, Kiesler und Strickroth 2024; Azaiz, Deckarm und Strickroth 2023; Roest, Keuning und Jeuring 2024). Die Ergebnisse der vorliegenden Untersuchung ergänzen diese Studien um die Beobachtung, dass Kontextanreicherung nicht nur die Fehlerhäufigkeit reduziert, sondern zugleich die Reproduzierbarkeit der Rückmeldungen verbessert und die inhaltliche Passung zur jeweiligen Aufgabe erhöhen kann.

Hinsichtlich **FF3.2** konnte erstmals systematisch belegt werden, dass LLMs wie `GPT-4` in der Lage sind, durch explizite Instruktion im Prompt zuverlässig spezifische Feedbacktypen zu generieren. Mit einer Übereinstimmung von 95 % zwischen gewünschtem und tatsächlich generiertem Feedbacktyp wurde ein sehr hoher Umsetzungsgrad erreicht. Gleichzeitig zeigen die Beobachtungen partieller Typüberschreitungen – insbesondere zwischen `KM` und `KH` sowie bei `KTC` – dass

eine vollständige Trennung zwischen bestimmten Feedbacktypen modellseitig nur eingeschränkt durchsetzbar ist. Dieses Ergebnis steht im Einklang mit der Annahme, dass LLMs stark von Mustern aus Trainingsdaten geprägt sind, in denen Fehlererklärung und Korrektur häufig kombiniert auftreten. Diese Beobachtung bestätigt die in der Literatur beschriebene Tendenz, dass in realen Rückmeldungen verschiedene Typen häufig kombiniert auftreten (Ditton 2014).

In beiden Untersuchungen zeigte sich, dass große Sprachmodelle grundsätzlich in der Lage sind, personalisierte Rückmeldungen zu erzeugen, die sich auf die individuelle Lösung der Lernenden beziehen. Häufig wurden konkrete Codeausschnitte aus den Einreichungen herangezogen, um Fehler zu erläutern oder Verbesserungsvorschläge zu formulieren. Diese Form der direkten Bezugnahme entspricht dem didaktischen Ideal, Feedback individuell und aufgabenbezogen zu gestalten, um eine hohe Relevanz und Wirksamkeit zu erzielen (Shute 2008). Vergleichbare Befunde liegen inzwischen auch für andere Modelle vor (Azaiz, Konrad und Strickroth 2025).

Im Vergleich zu klassischen automatisierten Feedbacksystemen eröffnet der Einsatz großer Sprachmodelle neue Möglichkeiten hinsichtlich Varianz, Differenziertheit und Personalisierung der Rückmeldungen. Durch gezielte Steuerung von Feedbacktypen und die Integration kontextueller Informationen kann die didaktische Qualität der generierten Hinweise deutlich verbessert werden. Gleichwohl bleibt die Notwendigkeit menschlicher Überprüfung bestehen: Besonders bei komplexen Aufgabenstellungen oder korrekten Lösungen treten weiterhin unzutreffende oder unpassende Rückmeldungen auf. Damit bestätigen die Ergebnisse die Schlussfolgerung anderer Studien, dass LLM-basierte Feedbacksysteme das Potenzial haben, die formative Unterstützung zu erweitern, jedoch nicht als vollständig autonome Instanzen, sondern als ergänzende Komponenten in einem didaktisch gerahmten Feedbackprozess zu verstehen sind.

6.5 Limitationen

Die Ergebnisse beider Untersuchungen sind vor dem Hintergrund mehrerer methodischer und konzeptueller Einschränkungen zu interpretieren. Dabei lassen sich einerseits gemeinsame, andererseits studienspezifische Limitationen identifizieren.

Gemeinsam ist beiden Studien zunächst die Abhängigkeit von den jeweils verwendeten Modellversionen (`GPT-3.5`, Stand März 2023, bzw. `GPT-4`, Stand März 2024). Da große Sprachmodelle in kurzen Entwicklungszyklen fortlaufend angepasst werden, ist davon auszugehen, dass sich Rückmeldungsqualität, Stabilität und Typenvielfalt zwischen Versionen oder Anbietern unterscheiden können. Die Ergebnisse sind daher nicht ohne Weiteres auf andere Modelle übertragbar. Ebenso beruhen beide Untersuchungen auf einem begrenzten Set von Programmieraufgaben, die zwar typische Fehlermuster und Basiskonzepte abdecken, jedoch nicht die gesamte Breite potenzieller Aufgabenformate repräsentieren. Rückmeldungen zu komplexeren, stärker offenen oder domänenspezifischen Aufgaben könnten daher abweichen.

Ein weitere Einschränkung betrifft die analytische Reduktion: Beide Analysen stützten sich auf theoriegeleitete Kategoriensysteme, die in Konsensverfahren angewendet wurden. Trotz Mehrfachkodierung bleibt ein Restrisiko von Interpretationsspielräumen bestehen, insbesondere bei Rückmeldungen, die mehrere Feedbacktypen überlagern und daher schwer eindeutig zuzuordnen sind. Auch die Reproduzierbarkeit der Ergebnisse ist eingeschränkt, da große Sprachmodelle stochastisch generieren und identische Eingaben zu unterschiedlichen Ausgaben führen können. Beide Untersuchungen bilden somit Momentaufnahmen ab, deren Ergebnisse nicht vollständig reproduzierbar sind. Schließlich ist zu betonen, dass beide Studien explorativen Charakter besitzen: Ziel war nicht die statistische Generalisierung, sondern die theoriegeleitete Charakterisierung typischer Muster. Quantitative Angaben dienen daher lediglich der Kontextualisierung und sind nicht als repräsentative Kennzahlen zu verstehen.

Über diese allgemeinen Einschränkungen hinaus ergeben sich studienspezifische Limitationen. Die erste Untersuchung fokussierte ausschließlich auf fehlerhaften Programmcode. Rückmeldungen zu korrekten oder teilweise richtigen Lösungen wurden nicht berücksichtigt, wodurch die Übertragbarkeit auf vollständige oder erfolgreiche Bearbeitungen eingeschränkt ist. Hinzu kommt, dass das Studiendesign bewusst auf *kontextarme Prompts* ohne Aufgabenstellung oder Lernkontext setzte, um die isolierte Feedbackfähigkeit des Modells zu prüfen. Diese Entscheidung ermöglicht eine kontrollierte Analyse, schränkt jedoch die Aussagekraft für realistischere Nutzungsszenarien mit didaktischer Rahmung ein. Auch die manuelle Auswahl der 33 untersuchten Submissions birgt trotz klarer Kriterien ein Restrisiko von Selektionsverzerrungen. Ebenso bleibt die Festlegung der 5 %-Schwelle für häufige Fehlertypen eine pragmatische Entscheidung.

Die zweite Untersuchung erweitert den Fokus, bringt jedoch eigene Einschränkungen mit sich. Im Gegensatz zur ersten Studie wurde hier eine stark kontextualisierte Promptgestaltung mit Aufgabenstellung, Musterlösung und Klassengerüst verwendet. Dies erlaubt eine realitätsnähere Analyse, begrenzt aber zugleich die Übertragbarkeit auf andere Promptformen oder Nutzungsszenarien. Auch wenn das Aufgabenset gegenüber der ersten Studie erweitert wurde, blieb es thematisch und konzeptionell eingegrenzt. Insbesondere Aufgaben mit höheren Komplexitätsstufen, wie objektorientierte Modellierung oder ereignisgesteuerte Anwendungen, wurden nicht einbezogen. Schließlich wurde pro Programmcode und Feedbacktyp jeweils nur eine Rückmeldung generiert, sodass die Varianz zwischen unterschiedlichen Generierungen unberücksichtigt blieb und keine Aussagen zur Stabilität der Modellantworten möglich sind.

Insgesamt liefern beide Studien komplementäre Einblicke in Potenziale und Grenzen LLM-generierter Rückmeldungen unter unterschiedlichen Rahmenbedingungen. Die Befunde sind als explorative Annäherung zu verstehen, die erste Hinweise auf relevante Einflussfaktoren und Muster bietet. Ihre Generalisierbarkeit über Modelle, Aufgabenformate und Nutzungskontexte hinaus ist jedoch eingeschränkt und sollte in weiterführenden Untersuchungen systematisch überprüft werden.

Kapitel 7

APFEL – ein konzeptueller Systementwurf für kontextsensitive Feedbackprozesse

Die bisherigen Untersuchungen und Modellierungen waren darauf ausgerichtet, Voraussetzungen dafür zu schaffen, Feedback-Strategien in digitalen Programmierlernumgebungen umsetzen zu können. In Kapitel 4 wurde ein Kategoriensystem für Feedback-Entscheidungen entwickelt, das Indikatoren für mögliche Rückmeldungen im Bearbeitungsprozess liefert und mögliche inhaltliche Ausgestaltungen beschreibt. Das in Kapitel 5 vorgestellte Rahmenmodell für Aufgaben und Antworten ermöglicht es, situative und instruktionale Faktoren sowie individuelle Faktoren in Form von Antwortklassen zu formalisieren und schafft damit die Voraussetzung, Feedback-Strategien kontext- und lernendenspezifisch sowie nachvollziehbar einzubetten. Die in Kapitel 6 dargestellten Ergebnisse verdeutlichen schließlich das Potenzial großer Sprachmodelle, Rückmeldungen inhaltlich auszugestalten, sofern ihre Generierung durch Strategien und Kontextinformationen gesteuert wird. Damit liegt ein Ansatz vor, Feedback-Typen nicht nur formal zu beschreiben, sondern auch automatisiert zu realisieren.

Diese Bausteine werden im folgenden Kapitel in einem konzeptuellen Systementwurf zusammengeführt, der eine Möglichkeit der Umsetzung von Feedback-Strategien in Programmierlernsystemen exemplarisch demonstriert. Berücksichtigt werden dabei der Zeitpunkt einer Rückmeldung (*Wann*), die inhaltliche Ausgestaltung (*Feedback-Typ*), situative und instruktionale Faktoren sowie individuelle Faktoren in Form von konkreten Ausgestaltungen der Lernendenantworten. Leitendes Strukturprinzip bildet der *externe Feedback-Loop* des ITFL-Modells: Ein Vergleich von IST- und SOLL-Zustand dient dabei als Ausgangspunkt, auf dessen Grundlage eine spezifische Feedback-Nachricht generiert wird. Ziel ist die funktionale Systemskizze eines Lernsystems für Einführungskurse im Programmieren, die demonstriert, wie kontextsensitive Feedbackprozesse sowohl skalierbar als auch nachvollziehbar realisiert werden können, sodass jede Intervention auf beobachtbare Artefakte und explizite Entscheidungsregeln zurückgeführt werden kann.

Der nachfolgend vorgestellte Systementwurf `APFEL` (kurz für: Adaptives Programmier-Feedback für E-Learning) versteht sich somit als konzeptuelle Brücke, welche die theoretischen und empirischen Ergebnisse der vorangegangenen Kapitel in ein *konzeptuelles Systemdesign* überführt und den Weg für die automatisierte Bereitstellung und systematische Erforschung adaptiver, kontextsensitiver Feedback-Strategien bereitet.

Teile dieses Kapitels wurden bereits vorab veröffentlicht in Lohr et al. (2024b). Für Details siehe Übersicht zu Vorabveröffentlichungen auf Seite VII.

7.1 Grundlegende Anforderungen und Designprinzipien

Die Konzeption des Systementwurfs orientiert sich an grundlegenden Anforderungen und Designprinzipien, die sich aus den theoretischen Vorarbeiten und den intendierten Einsatzszenarien ableiten lassen:

- **Orientierung an einem etablierten Modell für Feedback-Strategien**
  Als theoretische Grundlage dient das in Abschnitt 2.1.5 vorgestellte ITFL-Modell von Narciss (2018). Leitend ist hierbei der externe Regelkreis, der die Abfolge von Beobachtung, Diagnose, Rückmeldung und Evaluation strukturiert. Der Systementwurf soll diesen Zyklus technisch abbilden und damit die in den vorangegangenen Kapiteln entwickelten Bausteine in ein konsistentes Prozessmodell einbetten.

- **Transparenz der Entscheidungsprozesse**
  Ein zentrales Prinzip ist die Nachvollziehbarkeit aller Systementscheidungen. Feedback soll nicht rein datengetrieben entstehen, sondern regelbasiert auf Basis explizit definierter Kategorien und Entscheidungsregeln. Der jeweils bereitgestellte Kontext (z. B. Feedback-Typ) wird regelgeleitet ausgewählt und für die Feedbackgenerierung genutzt, sodass jede Rückmeldung auf ihre zugrunde liegenden Evidenzen zurückgeführt werden kann.

- **Skalierbarkeit über Aufgaben und Aufgabentypen hinweg**
  Damit das System nicht auf einzelne Aufgaben zugeschnitten bleibt, soll folgende klare Trennung stattfinden: Die regelbasierte Bereitstellung von Kontextinformationen erfolgt unabhängig von der eigentlichen Feedbackformulierung, die generativ ausgestaltet werden kann (z. B. durch Sprachmodelle). Auf diese Weise soll Skalierbarkeit über verschiedene Aufgaben und Aufgabentypen hinweg ermöglicht werden.

- **Integration in bestehende Programmierlernumgebungen**
  Das System soll funktional gängigen Programmierlernsystemen entsprechen und grundlegende Elemente wie Aufgabenstellungen, Code-Editor und Testfälle bereitstellen können. Für Lernende soll die Arbeit mit dem System daher ähnlich wie in einer üblichen Entwicklungsumgebung (IDE) wirken – mit dem entscheidenden Zusatz, dass kontextsensitives und didaktisch fundiertes Feedback verfügbar ist.

- **Anbindung von Lernendenmodellen**
  Um Rückmeldungen stärker an individuelle Faktoren anpassen zu können, ist eine Architektur erforderlich, die die Integration von Lernendenmodellen ermöglicht. So kann das Feedback nicht nur auf situative und aufgabenbezogene Kontexte, sondern auch auf den jeweiligen Lernstand der Studierenden abgestimmt werden.

- **Modularität und Austauschbarkeit von Komponenten**
  Die Architektur soll modular angelegt sein. Austausch und Erweiterung von Komponenten wie Diagnoseverfahren, Feedbackgeneratoren oder Lernendenmodelle soll möglich sein, ohne die Grundstruktur des Systems neu konzipieren zu müssen. Damit wird eine flexible Anpassung an unterschiedliche Forschungsszenarien und technologische Entwicklungen ermöglicht.

7.2 Zentrale Systemkomponenten

Die Architektur von `APFEL` orientiert sich am externen Regelkreis des ITFL-Modells. Dieser beschreibt Feedbackprozesse als zyklische Abfolge von *Messung*, *Vergleich*, *Entscheidung* und *Intervention*. Ausgangspunkt ist eine externe Messung, die den aktuellen Lern- und Leistungsstand erfasst und in Form eines IST-Werts abbildet. Dieser Wert wird mit einem vorab definierten

SOLL-Wert verglichen, der aus den Lernzielen, Lerninhalten und Aufgabenstellungen abgeleitet wird. Auf Basis dieses Vergleichs sowie weiterer Faktoren der Lehr-Lernsituation (z. B. Lernendenprofilen) trifft die externe Regeleinrichtung eine Entscheidung über eine geeignete Intervention. Diese äußert sich in einer *Stellgröße*, die im ITFL-Modell das externe Feedback darstellt. Das bereitgestellte Feedback wirkt anschließend über den internen Regelkreis auf die Lernaktivitäten zurück, wodurch ein neuer IST-Wert entsteht und der Zyklus erneut beginnt (siehe Abschnitt 2.1.5).

Ausgehend von diesem Mechanismus lassen sich zentrale Systemkomponenten für eine technische Umsetzung ableiten:

- **Diagnose-Komponente**
  Für die Erfassung des aktuellen Zustandes des Artefakts ist eine Diagnose-Komponente erforderlich, die Lernendenantworten systematisch analysiert und in Form eines IST-Werts repräsentiert. Dabei werden sowohl produktorientierte Evidenzen (z. B. Quellcode, Testergebnisse) als auch prozessorientierte Informationen (z. B. Zwischenschritte, Fehlermuster) berücksichtigt. Die Komponente muss den ermittelten IST-Wert mit dem aus Lernzielen, Aufgabenstellungen und erwarteten Lösungen abgeleiteten SOLL-Wert abgleichen, um Abweichungen präzise zu identifizieren und daraus potenzielle Ansatzpunkte für passendes Feedback abzuleiten.
- **Trigger-Komponente**
  Eine weitere Komponente übernimmt die Entscheidung, *ob* und *wann* eine Rückmeldung bereitgestellt werden soll. Sie stützt sich auf die in Kapitel 4 entwickelten Kategorien für Feedback-Entscheidungen und fungiert als Filter, der potenzielle Interventionspunkte im Bearbeitungsprozess identifiziert.
- **Kontextualisierer**
  Damit Feedback nicht isoliert, sondern in Bezug auf die Lehr-Lernsituation bereitgestellt wird, ist eine Komponente zur Kontextualisierung notwendig. Sie bündelt situative und instruktionale Faktoren (z. B. Lernziel, Aufgabenstellung) mit individuellen Faktoren (Zustand der Antwort, Lernendenprofil) und stellt diese Informationen für die Feedback-Generierung bereit.
- **Feedback-Generator**
  Ebenfalls erforderlich ist ein Generator, der auf Basis der Kontextinformationen und der getroffenen Regelentscheidungen die Rückmeldung erzeugt. Dies kann durch regelbasierte Formulierungen, durch generative Verfahren (z. B. Sprachmodelle) oder durch eine Kombination beider Ansätze erfolgen.

Abbildung 7.1 veranschaulicht die resultierenden Systemkomponenten von `APFEL` sowie deren funktionales Zusammenspiel.

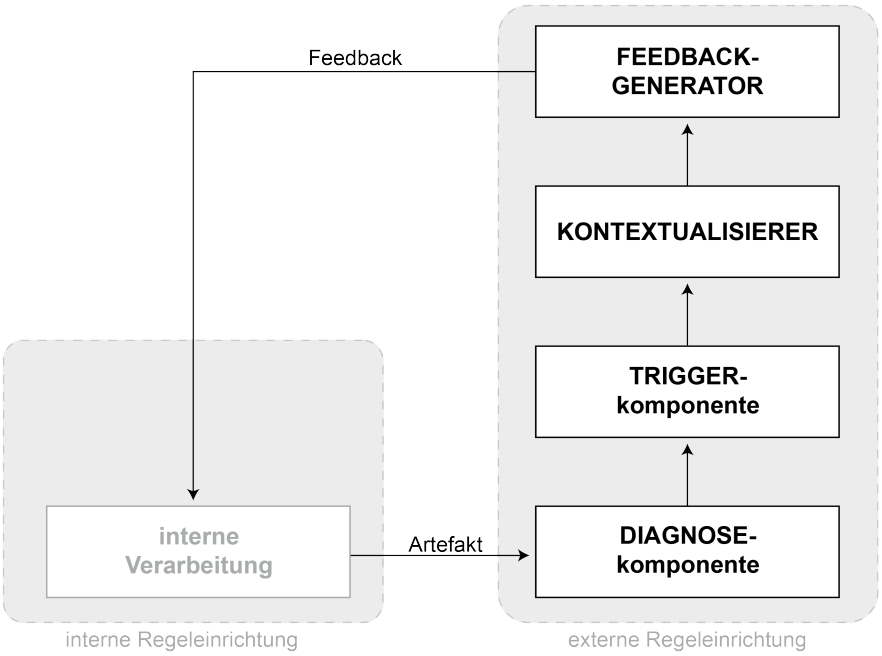


Abbildung 7.1: Systemkomponenten von `APFEL`, eigene Darstellung auf Basis des externen Regelkreises im ITFL-Modell (Narciss 2018)

7.3 Anforderungen an die Systemkomponenten

Aus den beschriebenen Systemkomponenten lassen sich konkrete Anforderungen ableiten, die eine Umsetzung der im ITFL-Modell verankerten Feedbackprozesse ermöglichen. Jede Komponente übernimmt dabei eine spezifische Funktion, deren Ausgestaltung in enger Verbindung zu den in den Kapitel 4, Kapitel 5 und Kapitel 6 gewonnenen Befunden steht.

Die *Diagnose-Komponente* bildet die Grundlage für alle weiteren Verarbeitungsschritte. Dafür sollte eine Möglichkeit bestehen, externe SOLL-Werte zu spezifizieren, die den erwünschten Zustand eines Lernartefakts definieren und damit als normative Referenz dienen, gegen die Lernendenantworten gemessen werden. Ebenso sollte die Diagnose-Komponente in der Lage sein, den aktuellen Zustand des Artefakts automatisiert als IST-Wert zu erfassen und diesen mit dem SOLL-Wert abzugleichen, um Abweichungen erkennbar zu machen und als Basis für mögliche Rückmeldungen nutzbar zu machen.

Die *Trigger-Komponente* hat die Aufgabe, Rückmeldungen an geeigneten Stellen im Bearbeitungsprozess auszulösen. Hierfür sollte sie in der Lage sein, spezifische Diskrepanzen zwischen IST- und SOLL-Wert zu erfassen und auf die in Kapitel 4 entwickelten Kategorien für Feedback-Entscheidungen abzubilden. Auf dieser Grundlage können potenzielle Interventionspunkte markiert und als Auslöser für den weiteren Prozess genutzt werden.

Für den *Kontextualisierer* sollte es möglich sein, situative und instruktionale Faktoren wie Lernziele, benötigtes Vorwissen oder mit der Aufgabe verbundene Konzepte in maschinenlesbarer Form zu repräsentieren und weiterzuverarbeiten (vgl. Kapitel 5). Damit wird angestrebt, dass Feedback aufgrund zu geringem Kontextes irreführend oder missverständlich ausfallen kann (vgl. Kapitel 6). Ebenso erscheint es erforderlich, die in Kapitel 4 rekonstruierten Kategorien zu Feedback-Typen einzubeziehen, sodass die inhaltliche Ausgestaltung der Feedback-Nachrichten systematisch berücksichtigt werden kann. Darüber hinaus sollte es möglich sein, aus bestimmten

Kontextkonstellationen konkrete Feedback-Entscheidungen abzuleiten und diese als Grundlage für Feedback-Strategien zu nutzen. Schließlich ist eine Schnittstelle zu Lernendenmodellen zweckmäßig, um kontextbezogene Informationen mit individuellen Faktoren zu verknüpfen und Rückmeldungen perspektivisch an persönliche Lernvoraussetzungen anzupassen.

Der *Feedback-Generator* schließlich ist für die konkrete Formulierung und Bereitstellung der konkreten Feedback-Nachrichten vorgesehen. Hierfür sollte er in der Lage sein, auf Basis des bereitgestellten Kontexts passendes Feedback zu erzeugen, das inhaltlich konsistent zu den vorherigen Verarbeitungsschritten ist. Insbesondere bei der Nutzung großer Sprachmodelle ist es erforderlich, den relevanten Kontext dynamisch in Prompts überführen zu können und durch geeignete Instruktionen anzureichern. Final wird eine Schnittstelle zur Lernplattform benötigt, damit das generierte Feedback in geeigneter Form – etwa direkt im Code-Editor oder als Teil des Aufgabenfeedbacks – an die Lernenden ausgegeben werden kann.

7.4 Umsetzung der Systemkomponenten

Die in Abschnitt 7.3 beschriebenen Anforderungen bilden den Rahmen dafür, wie der externe Feedback-Regelkreis im Prototyp `APFEL` technisch umgesetzt werden kann. In den nachfolgenden Abschnitt wird auf die konkreten Ansätze eingegangen, mit denen die einzelnen Systemkomponenten realisiert werden können. Ziel ist es zu zeigen, welche technologischen Verfahren, Werkzeuge und Datenrepräsentationen jeweils zum Einsatz kommen können und wie diese zusammenspielen, um den Regelkreis praktisch funktionsfähig zu machen. Dabei werden die vier Kernkomponenten – Diagnose-Komponente, Trigger-Komponente, Kontextualisierer und Feedback-Generator – jeweils gesondert betrachtet und anhand prototypischer Lösungen erläutert.

7.4.1 Diagnose-Komponente: Abbildung von SOLL- und IST-Werten

Die in Kapitel 5 vorgestellten Antwortklassen bilden die konzeptuelle Grundlage für die Diagnose-Komponente. Im Prototyp `APFEL` dienen sie als zentrales Instrument, um Lernendenlösungen einem definierten SOLL-Zustand gegenüberzustellen und den aktuellen IST-Zustand zu bestimmen. Antwortklassen fungieren dabei als Referenzen, die durch geeignete Analysen diagnostisch überprüft werden. Der SOLL-Zustand einer Aufgabe ist dabei durch eine konkrete Kombination von Antwortklassen beschrieben, die vollständig erfüllt sein muss, während zugleich bestimmte andere Antwortklassen explizit *nicht* erfüllt sein dürfen. Erst das Zusammenspiel dieser positiven und negativen Kriterien bildet die normative Referenz, an der Lernendenlösungen gemessen werden.

Die technische Umsetzung erfolgt in `APFEL` mit einer Kombination aus dynamischer und statischer Code-Analyse, für die Komponenten des E-Assessment-Systems `JACK` zum Einsatz kommen (Striewe 2016). Dank seiner modularen Architektur erlaubt `JACK` deren Einbindung unabhängig von anderen Systemkomponenten:

Die dynamische Analyse erfolgt durch Unit-Tests, die für jede Antwortklasse spezifische Testkonstellationen abdecken. Unter Unit-Tests sind hierbei gezielte Eingabe-Ausgabe-Prüfungen zu verstehen, wie sie aus der Softwareentwicklung bekannt sind: Ein Programmfragment wird mit vordefinierten Eingaben ausgeführt und das Ergebnis mit den erwarteten Ausgaben verglichen. Auf diese Weise lassen sich beispielsweise Korrektheitseigenschaften überprüfen oder auch Laufzeiteigenschaften wie Endlosschleifen oder Ausnahmen sichtbar machen. Unit-Tests bilden somit die Grundlage, um produktorientierte Evidenzen systematisch zu erfassen und mit den Erwartungen abzugleichen.

Ergänzend werden mittels statischer Code-Analyse Eigenschaften des Quellcodes überprüft, die sich nicht unmittelbar durch Ausführungstests erfassen lassen. Hierfür bietet das `JACK`-System die Möglichkeit, über die Graphabfragesprache *GreQL* beliebige Abfragen über Graphenstrukturen durchzuführen, sodass syntaktische und strukturelle Eigenschaften direkt auf dem abstrakten Syntaxbaum analysiert werden können. Mit GreQL können beispielsweise Regeln formuliert werden, die nach der Verwendung bestimmter Sprachkonstrukte suchen, die Einhaltung bestimmter Programmierkonventionen kontrollieren oder charakteristische Fehlermuster identifizieren. Eine Übersicht typischer Abfrageoperationen bietet die GReQL-Referenzkarte[1], die etwa Operatoren für Pfadabfragen, Mustererkennung im Syntaxbaum oder die Selektion bestimmter Knotentypen auflistet. Damit lassen sich feinkörnige Kriterien für Antwortklassen operationalisieren, die über reine Funktionalitätstests hinausgehen.

Auf dieser Grundlage entsteht ein Mechanismus, bei dem jede Antwortklasse durch eine definierte Kombination aus Unit-Testfällen sowie GreQL-Abfragen charakterisiert wird. Das Ergebnis ist eine Menge von Antwortklassen, die gemeinsam den IST-Zustand der Lösung beschreiben und in Relation zum definierten SOLL-Zustand gesetzt werden kann. Dieser Vergleich bildet schließlich die Grundlage für den weiteren Verlauf im externen Feedback-Regelkreis.

7.4.2 Trigger-Komponente: Identifikation von Interventionspunkten

Während die Diagnose-Komponente den aktuellen IST-Zustand einer Lösung bestimmt, übernimmt die Trigger-Komponente die Aufgabe, potenzielle Interventionspunkte zu erkennen. Ziel ist es, zu bestimmen, ob an einer bestimmten Stelle im Bearbeitungsprozess Feedback bereitgestellt werden soll. Die Umsetzung erfolgt regelbasiert, sodass die Auslöselogik jederzeit nachvollziehbar und transparent bleibt. Jede Regel verweist dabei explizit auf diagnostische Evidenzen (Antwortklassen, IST-/SOLL-Vergleiche, Lernendenaktionen) und kann so überprüft oder angepasst werden.

Die Grundlage bilden die in Kapitel 4 rekonstruierten Kategorien für Feedback-Entscheidungen, die hier in formale Auslösekriterien (*Trigger*) übersetzt werden. Produktbezogene situative Faktoren (vgl. Abschnitt 4.3.2.1) wie **SYNE**, **TYPE** oder **LOGE** lassen sich direkt über die Zuordnung zu bestimmten Antwortklassen diagnostizieren (siehe Abschnitt 7.4.1). Ergänzend können prozessbezogene situative Faktoren durch den Vergleich von IST- und SOLL-Zustand bestimmt werden. So entspricht etwa die Kategorie **START** der initialen Antwortklassenkombination zu Beginn der Bearbeitung, wenn alle Checks gegen den unveränderten Programmcode ausgeführt werden. Die Kategorie **COMPLETE** liegt vor, sobald der IST-Zustand vollständig mit dem definierten SOLL-Zustand übereinstimmt.

Für die Kategorie **STEPBACK** ist zusätzlich eine Historie der Antwortklassenkombinationen erforderlich. Hierzu wird bei jeder Abgabe die aktuelle Kombination gespeichert, sodass Veränderungen im Bearbeitungsverlauf sichtbar werden. Ein Rückschritt kann dann regelbasiert diagnostiziert werden, wenn eine Lösung auf eine zuvor bereits diagnostizierte Kombination von Antwortklassen „zurückfällt". Auch die Kategorie **T&E** (*Trial & Error*) lässt sich mit Hilfe dieser Historie erfassen. Ein typisches Indiz für `T&E` ist das Auftreten zyklischer Muster in den Antwortklassenkombinationen: Wenn Lernende mehrfach zwischen denselben oder sehr ähnlichen Antwortklassen wechseln, ohne eine stabile Annäherung an den SOLL-Zustand zu erreichen, kann dies als Indiz auf Ausprobieren ohne systematische Strategie interpretiert werden. Diagnostisch lässt sich dies durch Regeln abbilden, die wiederholte Sequenzen in der Folge von Antwortklassenkombinationen erkennen und als potenziellen Interventionspunkt markieren.

[1] `https://wiki.uni-due.de/jack/images/4/46/GReQL-Referencecard.pdf`

Weitere prozessbezogene situative Faktoren setzen zusätzliche Informationen aus der Lernplattform oder aus einem Lernendenmodell voraus. Ein triviales Beispiel ist die Kategorie **REQUEST**, die immer dann vorliegt, wenn eine konkrete Anfrage nach Feedback durch die Lernenden ausgelöst wird (z. B. durch Verwendung eines HELP-Buttons). Diese Information ergibt sich unmittelbar aus dem Interaktionsprotokoll der Plattform und benötigt keine weiteren Analysen. Komplexer gestaltet sich die Kategorie **TRUST**, die nicht aus einer einzelnen Antwort oder Interaktion abgeleitet werden kann, sondern Informationen über den Lernenden im Zeitverlauf erfordert. `TRUST` bezieht sich auf die Wahrscheinlichkeit, dass ein Lernender auf Basis seines bisherigen Bearbeitungsverhaltens bzw. seiner Lernhistorie den nächsten Bearbeitungsschritt erfolgreich bewältigen wird oder nicht. Damit diese Einschätzung möglich ist, muss das System Daten über den Lernenden in geeigneter Form speichern und abfragbar machen. Dies kann im Rahmen eines expliziten (persistenten) Lernendenmodells erfolgen, in dem Bearbeitungsschritte, Antwortmuster und Verlaufskontexte kontinuierlich dokumentiert werden oder in Form eines sogenannten *Ad-hoc-Lernendenmodell*, das eine temporäre, aufgabenspezifische Repräsentation des bisherigen Bearbeitungsprozesses ermöglicht, die aus den unmittelbar verfügbaren Interaktionsdaten gewonnen wird. Schon diese reduzierte Form der Modellierung erlaubt es, in späteren Schritten Interventionspunkte wie `TRUST` zu identifizieren – etwa wenn sich aus der Abfolge der Antwortklassenkombinationen oder den wiederholt auftretenden Fehlermustern ergibt, dass ein selbständiger Fortschritt mit hoher Wahrscheinlichkeit nicht mehr zu erwarten ist. Die konzeptionelle Grundlage und konkrete technische Umsetzung des Lernendenmodells wird im nachfolgenden Abschnitt zum Kontextualisierer näher beschrieben.

Zusammenfassend verarbeitet die Trigger-Komponente also Informationen aus dem aktuellen IST-Zustand (Antwortklassen), aus Interaktionen in der Lernplattform sowie aus Daten eines Lernendenmodells und markiert daraus abgeleitet mögliche Interventionspunkte. Diese Interventionspunkte können auf unterschiedliche Weise genutzt werden: Der Bearbeitungsprozess kann von der Lernplattform aktiv unterbrochen werden, um direkt Feedback bereitzustellen. Alternativ ist auch ein weniger aufdringlicher Hinweis denkbar, dass eine bestimmte Form von Unterstützung verfügbar ist, falls diese gewünscht wird oder es können Hinweise gegeben werden, dass bestimmte Aktionen – wie etwa das Ausführen des Programms – weiterhelfen könnten. Ebenso kann die Bereitstellung von Feedback auch bewusst eingeschränkt werden, wenn Indizien vorliegen, dass Lernende den nächsten Schritt voraussichtlich selbst bewältigen können. In solchen Fällen kann die Hilfestellung bewusst (noch) zurückgehalten werden, um selbstständiges Problemlösen zu fördern und Über-Unterstützung zu vermeiden.

7.4.3 Kontextualisierer

Die Aufgabe des Kontextualisierers besteht darin, die situativen und instruktionalen Faktoren einer Lernsituation maschinenlesbar bereitzustellen und mit diagnostischen Informationen zu verknüpfen. Auf diese Weise wird sichergestellt, dass Rückmeldungen nicht isoliert entstehen, sondern in Bezug auf Lernziele, Vorwissen und die mit einer Aufgabe verbundenen Konzepte interpretiert werden können.

Für die technische Umsetzung greift `APFEL` auf den adaptiven Lernassistenten ALeA zurück, der bereits über ein Domänen- und Lernendenmodell verfügt und Annotationen im Sinne des in Kapitel 5 eingeführten Y-Modells unterstützt (Betzendahl, Kohlhase und Müller 2024). Grundlage bildet die semantische Annotation von Kursmaterialien, durch die Konzepte, Abhängigkeiten und kognitive Anforderungen explizit markiert und in einer maschinenlesbaren Form zugänglich gemacht werden.
Die Annotation erfolgt in ALeA mit Hilfe von sTeX, einer Sammlung von LaTeX-Makropaketen für semantisches Markup (Kohlhase 2008). sTeX ermöglicht es, den inhaltlichen Gehalt von Lernmaterialien direkt in den Quelltexten zu kennzeichnen, etwa indem Begriffe, Definitionen oder

Aufgabenstellungen mit semantischen Metadaten versehen werden. Auf diese Weise werden inhaltliche Strukturen, die in herkömmlichen Lehrmaterialien nur implizit vorhanden sind, explizit gekennzeichnet und in einer für maschinelle Verarbeitung geeigneten Form repräsentiert. Die mit SᴛEX erzeugten semantischen Strukturen werden anschließend ins FTML-Format überführt, ein foundation-unabhängiges HTML-basiertes Repräsentationsformat für die Repräsentation semantisch annotierter Dokumentfragmente (Müller 2026). FLAMS fungiert dabei als universelle Zwischenschicht, in der Konzepte, Theorien und ihre Relationen in einem standardisierten Format gespeichert und über standardisierte Schnittstellen abgefragt werden können. Durch diese Abbildung werden Abhängigkeiten zwischen Konzepten explizit modelliert und können im Feedbackprozess berücksichtigt werden.

Damit entsteht eine Infrastruktur, in der Lernziele, Vorwissen und Aufgabenkontexte nicht nur textuell beschrieben, sondern formalisiert und programmatisch abrufbar sind. Für die Kontextualisierungs-Komponente von `APFEL` bedeutet dies, dass relevante Faktoren über eine API abgefragt und mit den Informationen der Diagnose- und Trigger-Komponente kombiniert werden können. So können konkrete Kontextkonstellationen identifiziert werden, um eine Auswahl und Ausgestaltung von Feedback-Strategien zu steuern.

Für die Modellierung von Lernenden im Rahmen der Feedbackprozesse wird in `APFEL` das Lernendenmodell von ALeA genutzt (Betzendahl, Kohlhase und Müller 2024). Dieses erfasst den Lernstand von Lernenden auf der Ebene einzelner Konzepte (analog zu *KnoCs* in Kapitel 5) und verknüpft diese mit unterschiedlichen kognitiven Dimensionen nach der AKT (siehe Abschnitt 5.1.2.2). Für jedes Paar aus Konzept und Dimension wird eine Wahrscheinlichkeit $p \in [0, 1]$ gespeichert, die den vermuteten „Kompetenzgrad" des Lernenden widerspiegelt.
Das Lernendenmodell wird in ALeA getrennt vom eigentlichen Domänenmodell verwaltet und ist über eine API abfragbar. So können zu jedem Zeitpunkt Werte wie „Lernender beherrscht Konzept C auf Dimension D mit Wahrscheinlichkeit p" in den Feedbackprozess eingebunden werden. Diese Abfragen ermöglichen es Kategorien, die eine Einschätzung des Lernstands oder des voraussichtlichen Lernfortschritts erfordern (z. B. `PROG`) zu berücksichtigen. Eine Besonderheit ist, dass dieses Modell nicht zwingend dauerhaft und umfassend ausgeprägt sein muss. Neben einem persistenten Lernendenmodell erlaubt die Architektur auch den Einsatz von *Ad-hoc-Modellen*, die nur auf Basis der unmittelbar verfügbaren Bearbeitungs- und Interaktionsdaten generiert werden. Damit können auch kurzfristige Verläufe, etwa wiederkehrende Fehlermuster oder Rückschritte im Bearbeitungsprozess, als Grundlage für Feedback-Entscheidungen genutzt werden. Durch die Kopplung von `APFEL` mit dem Lernendenmodell von ALeA wird somit eine Schnittstelle geschaffen, über die Lernendenprofile in den externen Feedback-Regelkreis eingebunden werden können. Auf diese Weise können Interventionsentscheidungen nicht nur auf Antwortklassen und situative Faktoren gestützt, sondern auch an individuelle Kompetenzstände angepasst werden.

7.4.4 Feedback-Generator

Der Feedback-Generator ist technologieoffen konzipiert: Grundsätzlich könnten auch Template-Engines, regelbasierte Textbausteine oder manuell formuliertes Feedback für bestimmte Kontext-Kombinationen eingesetzt werden. Im Prototyp `APFEL` werden jedoch große Sprachmodelle genutzt, um Rückmeldungen in natürlicher Sprache zu formulieren. Dabei baut die Umsetzung auf den in Kapitel 6 entwickelten Ansätzen auf. Wichtig ist, dass der Generator nicht darüber entscheidet, *welcher* Feedback-Typ erzeugt wird – dieser liegt bereits durch die Regelentscheidung der Trigger-Komponente fest. Aufgabe des Generators ist es vielmehr, den gewählten Typ sprachlich auszugestalten und an den spezifischen Aufgaben- und Lernkontext anzupassen.

Die Generierung erfolgt auf Basis von Prompt-Vorlagen, die mit Platzhaltern für zentrale Informationen wie etwa Feedback-Typ, Muster-Lösung, Aufgabenbeschreibung, Beispiele oder Notationen versehen werden können. Diese Platzhalter werden im Generationsprozess mit den von Diagnose- und Kontextualisierungs-Komponente bereitgestellten Informationen befüllt. Da die Aufgabenstellung und das Kursmaterial semantisch annotiert vorliegen (vgl. Abschnitt 7.4.3), können relevante Definitionen, Beispiele oder Konventionen programmgesteuert in den Prompt integriert werden. Dadurch wird sichergestellt, dass Rückmeldungen konsistent mit dem im Kurs etablierten Begriffssystem und den verwendeten Darstellungsformen sind. Dies ist insbesondere für Feedback-Typen wie *Knowledge about Concepts* relevant, die sich explizit auf fachliche Konzepte beziehen und daher mit den gewählten Formulierungen und Notationen des Kurs-Kontexts übereinstimmen sollten.

Um diese Kontextintegration systematisch zu ermöglichen, wird ein Retrieval-Augmented-Generation-Ansatz (RAG) genutzt (Lewis et al. 2020). Hierbei können aus den annotierten Kursmaterialien gezielt Wissenselemente wie Definitionen, Beispielaufgaben oder häufige Fehlkonzepte abgerufen werden und in den Prompt eingebettet (Jacobs und Jaschke 2024; Lohr et al. 2025b). Die Generierung erfolgt somit nicht isoliert, sondern auf Grundlage evidenzbasierter Informationen aus dem Lehrmaterial. Dies reduziert die Wahrscheinlichkeit, dass Rückmeldungen widersprüchlich, unpräzise oder kursfremd ausfallen, und stärkt zugleich die Nachvollziehbarkeit der Ergebnisse. Schließlich wird jede Generierung protokolliert, indem die zugrunde liegenden Regeln, die genutzten Kontextinformationen und die erzeugte Rückmeldung gespeichert werden. Auf diese Weise kann jede bereitgestellte Rückmeldung auf die verwendeten Artefakte und Entscheidungsregeln zurückgeführt werden. Die Ausgabe erfolgt über eine Schnittstelle zur Lernplattform und kann je nach Anforderung als Text, HTML oder in anderen Formaten bereitgestellt werden.

Damit verbindet der Feedback-Generator eine regelgeleitete Festlegung des *Was* mit einer kontextsensitiven, durch RAG gestützten Ausgestaltung des *Wie*. Der Generationsprozess bleibt so kontrollierbar, skalierbar und nachvollziehbar.

7.5 Fallbeispiel (fiktiv)

Um das Zusammenspiel der in den vorangegangenen Abschnitten beschriebenen Systemkomponenten zu illustrieren, wird im Folgenden ein fiktives Fallbeispiel durchgespielt. Als Beispielaufgabe dient die Implementierungsaufgabe zur Berechnung der Fibonacci-Zahlen, die bereits in Kapitel 4 und Kapitel 6 eingesetzt wurde. Sie eignet sich besonders, da aus den dortigen Untersuchungen typische Fehlermuster dokumentiert sind, die hier aufgegriffen werden können. Zudem liegen aus Kapitel 6 bereits generierte Rückmeldungen zu dieser Aufgabe vor, die in das Beispiel integriert werden können.

Da empirische Studien zur Wirksamkeit spezifischer Feedback-Strategien in diesem Kontext bislang nicht vorliegen, stützt sich das Fallbeispiel auf *hypothetische Strategien*, die hier rein demonstrativ eingesetzt werden. Diese Strategien sind nicht als endgültige Empfehlungen zu verstehen, sondern dienen ausschließlich der Veranschaulichung, wie sich Feedback-Entscheidungen technisch im System abbilden lassen. Forschenden steht es selbstverständlich frei, diese exemplarischen Annahmen durch eigene Studien zu bestätigen, zu modifizieren oder zu widerlegen.

7.5.1 Beschreibung des Szenarios

Das fiktive Szenario ist in einer typischen Einführungsveranstaltung zur Programmierung an einer Hochschule verortet, die von *Prof. Dr. Clara Code* geleitet wird. Die Programmiersprache der Veranstaltung ist Java. Bereits in den ersten Wochen des Semesters werden grundlegende

Konzepte wie Datentypen, Kontrollstrukturen und Felder eingeführt. Um diese Inhalte zu festigen, stellt sie den Studierenden im Rahmen der dritten Übung eine Programmieraufgabe bereit, die den Umgang mit Feldern und Wiederholungen praktisch erproben lässt.

Die Aufgabe besteht darin, eine Methode `minSearch` zu implementieren, die ein Feld von Ganzzahlen als Eingabe erhält und das kleinste enthaltene Element zurückgibt. Sie ist so gewählt, dass sie einerseits den typischen Schwierigkeitsgrad für ein Übungsszenario in dieser frühen Phase des Kurses aufweist und andererseits eine Reihe charakteristischer Fehlermuster provozieren kann.

Prof. Dr. Clara Code entscheidet, die Aufgabe im Rahmen von `APFEL` bereitzustellen, sodass die Studierenden sie selbstständig im Übungssystem bearbeiten können. Damit die Aufgabe in `APFEL` nutzbar ist, sind mehrere vorbereitende Schritte notwendig:

- Die Aufgabenstellung muss semantisch annotiert werden, sodass Lernziele, Vorwissensanforderungen und zentrale Konzepte (z. B. *Feld*, *Minimum*, *for-loop*) maschinenlesbar vorliegen und im Kontextualisierer genutzt werden können.
- Es müssen passende *Antwortklassen* entwickelt oder aus einem bestehenden Katalog ausgewählt werden.
- Die Antwortklassen müssen durch konkrete statische und dynamische Analysen operationalisiert werden.
- Optional können neben den allgemeinen, in der Forschung beschriebenen Feedback-Strategien, die bereits in `APFEL` vorgesehen sind, auch zusätzliche individuelle Strategien für das konkrete Beispiel festgelegt werden.

Auf dieser Grundlage kann die Aufgabe in `APFEL` eingebunden werden und steht den Studierenden als selbstständige Übung zur Verfügung. Im weiteren Verlauf wird nun in einem fiktiven Szenario demonstriert, wie die Bereitstellung der Aufgabe in `APFEL` konkret ablaufen würde und die Systemkomponenten im Zusammenspiel an diesem Fallbeispiel arbeiten.

7.5.2 Vorbereitung der Aufgabe: Bereitstellung und Annotation

Bevor die Systemkomponenten von `APFEL` wirksam werden, sind zunächst einige vorbereitende Schritte erforderlich. Clara Code entscheidet sich für die Aufgabe *minSearch*, die in einem Lehrbuch gefunden hat und die sich für den aktuellen Kurs gut eignet. In ihrer „Urform" sieht die Aufgabenstellung wie in Abbildung 7.2 dargestellt aus: eine kurze Instruktion in Textform kombiniert mit einem leeren Klassenrumpf, in den die Lernenden ihre Lösung einfügen sollen.

minSearch

Gegeben sei der folgende (leere) Klassenrumpf `Homework3` in Java:

```
public class Homework3 {
    // Code here
}
```

Implementieren Sie eine Methode `minSearch`, die ein Feld (`int[]`) von Ganzzahlen als Eingabe erhält und das kleinste darin enthaltene Element zurückgibt.

Abbildung 7.2: Aufgabenstellung *minSearch* (deutsche Fassung, Urform)

Damit die Aufgabe im System `APFEL` nutzbar ist, werden zusätzliche Artefakte bereitgestellt. Dazu gehört zunächst der sogenannte *Skeleton Code*, der den initialen Zustand der Quellcodedatei abbildet und den Studierenden beim Öffnen des Editors angezeigt wird. Dieser Skeleton wird als

separate Datei (`skeleton.java`) in das System hochgeladen und entspricht inhaltlich dem in Abbildung 7.2 dargestellten Klassenrumpf.

Darüber hinaus hinterlegt Clara Code eine spezifische *Musterlösung*, die später sowohl in der Diagnose-Einheit (z. B. als Referenz für Unit-Tests) als auch im Feedback-Generator (z. B. zur Integration in Prompts) verwendet werden kann. Diese Musterlösung ist nicht für die Studierenden sichtbar, sondern dient ausschließlich der internen Verarbeitung im System.

Im nächsten Schritt wird die Aufgabenstellung aus ihrer textuellen Urform in eine LaTeX/sTeX-Repräsentation überführt, um sie für Annotationen und die maschinenlesbare Weiterverarbeitung zugänglich zu machen.

Semantische Annotation der Aufgabenstellung

Nachdem die Aufgabenstellung in eine LaTeX-Repräsentation überführt wurde (vgl. Abbildung 7.3), folgt im nächsten Schritt die semantische Annotation. Diese bildet die Grundlage dafür, dass zentrale Konzepte der Aufgabe maschinenlesbar vorliegen und im Kontextualisierer von `APFEL` genutzt werden können. Durch die Verwendung der `sproblem`-Umgebung wird die Aufgabenstellung zugleich eindeutig als solche markiert und damit systemseitig verarbeitbar. Wie bereits in Kapitel 5 diskutiert, ist die Annotation der Aufgabenstellung eine notwendige Voraussetzung, damit Aufgaben in adaptiven Feedbacksystemen formal repräsentiert und kontextsensitiv verarbeitet werden können.

Da `APFEL` auf `write-code`-Aufgaben ausgelegt ist (vgl. Ruf, Berges und Hubwieser (2015)), sind Aufgabenformat und Repräsentationsformate in diesem Kontext bereits festgelegt. Die instruktive Struktur ergibt sich aus den in sTeX vorgesehenen Umgebungen: Neben der `sproblem`-Umgebung für die Aufgabenstellung können beispielsweise zusätzliche Umgebungen wie `sexample` genutzt werden, um Beispiele oder Testfälle formal anzugeben. So ließe sich etwa ein Beispiel-Feld `[1, 3, 5, 7]` mit der erwarteten Ausgabe `1` hinzufügen. Im vorliegenden Fall wird auf eine solche Ergänzung verzichtet, da die Aufgabe bewusst offen gehalten ist und ohne vorgegebene Beispiele auskommen soll.

```
\begin{sproblem}

    Gegeben sei der folgende (leere) Klassenrumpf \texttt{Homework3}

    \begin{listing}[language=Java]
        public class Homework3 {
            // Code here
        }
    \end{listing}

    Implementieren Sie eine Methode \texttt{minSearch}, die ein Feld
    (\texttt{int[]}) von Ganzzahlen als Eingabe er\"ahlt und das kleinste
    darin enthaltene Element zur\"uckgibt.

\end{sproblem}
```

Abbildung 7.3: Aufgabenstellung zu `minSearch` in `sproblem`-Umgebung ohne Annotationen

Im nächsten Schritt sollen nun die in der Aufgabe vorkommenden Konzepte annotiert und mit dem Domänenmodell von ALEA verknüpft werden. Im Szenario übernimmt *Prof. Dr. Clara Code* die Annotation. Zunächst kennzeichnet sie die Konzepte, die explizit in der Aufgabenbeschreibung genannt sind: *Methode*, *Feld*, *Ganzzahl*. Darüber hinaus macht sie implizite Konzepte sichtbar, die für das Verständnis und die Bearbeitung der Aufgabe aus ihrer Sicht notwendig

sind. So verweist etwa "das kleinste darin enthaltene Element" auf das Konzept des *Minimums*. Dieses ist zwar eher in der Domäne der Mathematik zu verorten und nicht originär der Programmierdomäne zugeordnet, wird aber dennoch aufgenommen, da es für die Lösung zentral ist und im Domänenmodell von ALeA bereits existiert. Ebenso wird das Konzept *Parameter* annotiert, das in der Aufgabenbeschreibung als "Eingabe" bezeichnet ist.

Zum Annotieren der Konzepte wird das STEX-Makro `\sr` (*semantic reference*) genutzt, mit dem sich Konzepte referenzieren und zugleich der anzuzeigende Text spezifizieren lassen. Damit die Konzepte verfügbar sind, importiert Clara Code die passenden Module aus dem Repository von ALeA (siehe Zeile 1-3 in Abbildung 7.4). In diesem Fall wählt sie unter anderem die Module für Arithmetik und objektorientierte Programmierung, da diese die benötigten Konzepte enthalten. Für den Begriff *Klassenrumpf* findet sie kein passendes Konzept im Repository und legt daher ein neues an.

Neben den Konzeptannotationen ergänzt Clara Code auch Angaben zu Vorwissen und Lernzielen. Mit dem Makro `\precondition` markiert sie, welche Vorkenntnisse auf welchen Dimensionen der AKT (vgl. Abschnitt 5.1.2.2) aus ihrer Sicht notwendig sind. So annotiert sie beispielsweise, dass das Konzept *Feld* verstanden und grundlegende Kenntnisse zur Kontrollstruktur Wiederholung erinnert werden sollten. Über das Makro `\objective` legt sie fest worauf die Aufgabe primär abzielt: Studierende sollen das Konzept der Wiederholung praktisch *anwenden*. Da die Aufgabenstellung offen lässt, welche Art von Kontrollstruktur zu verwenden ist, annotiert sie bewusst nur das generische Konzept *loop* und nicht etwa *for-loop*.

Das Ergebnis ist eine semantisch angereicherte Aufgabenstellung, wie sie in Abbildung 7.4 dargestellt ist. Durch diese Annotation wird sichergestellt, dass die inhaltlichen Bezüge der Aufgabe – sowohl aus der Informatik als auch aus angrenzenden Domänen – formal repräsentiert und in weiteren Verarbeitungsschritten von `APFEL` berücksichtigt werden können.

```
1 \usemodule[smglom/arithmetics]{mod?minmax}
2 \usemodule[smglom/computing]{mod?oop}
3 \usemodule[ccode/java]{mod?class-body}
4
5 \begin{sproblem}
6     \precondition{remember}{java-syntax}
7     \precondition{remember}{loop}
8     \precondition{understand}{minmax?minimum}
9     \precondition{understand}{array}
10    \objective{apply}{loop}
11
12    Gegeben sei der folgende (leere) \sr{ccode?class-body}{Klassenrumpf} \texttt{Homework3}:
13
14    \begin{listing}[language=Java]
15        public class Homework3 {
16            // Code here
17        }
18    \end{listing}
19
20    \sr{implement}{Implementieren} Sie eine \sr{oop?method}{Methode} \texttt{minSearch}, die ein
21    \sr{array}{Feld} von \sr{integer}{Ganzzahlen} als \sr{parameter}{Eingabe} er\"ahlt und
22    \sr{minmax?minimum}{das kleinste darin enthaltene Element} zur\"uckgibt.
23 \end{sproblem}
```

Abbildung 7.4: Aufgabenstellung mit annotierten Konzepten, Vorbedingungen und Lernzielen

Antwortklassen und Testfälle

Im nächsten Schritt wählt *Prof. Dr. Clara Code* aus dem Repository von Antwortklassen diejenigen aus, die für die Aufgabenstellung `minSearch` relevant erscheinen. Dabei greift sie auf Erfahrungen aus der letzten Kohorte zurück, in der eine ähnliche Aufgabe bereits eingesetzt wurde. Als Ausgangspunkt entscheidet sie sich für folgende Antwortklassen:

AC_0: Methode `<Methodenname>` ist funktional korrekt.

AC_1: Methode `<Methodenname>` liefert für Felder mit positiven Zahlen korrektes Ergebnis.

AC_2: Methode `<Methodenname>` liefert für Felder mit negativen Zahlen korrektes Ergebnis.

AC_3: Methode `<Methodenname>` behandelt den Spezialfall eines leeren Feldes.

AC_4: In Klasse `<Klassenname>` existiert eine Methode mit Bezeichner `<Methodenname>`.

Dabei fungiert AC_0 als Oberklasse für die spezifischeren Antwortklassen AC_1 – AC_3: Wird AC_1, AC_2 und AC_3 jeweils erfüllt, gilt die Lösung insgesamt als funktional korrekt. Für diese Antwortklassen existiert bereits ein Set an Unit-Tests, mit denen die Methode anhand verschiedener Felder geprüft wird. Um Variationen gegenüber der vorherigen Kohorte zu schaffen, passt Clara Code lediglich einzelne Eingabewerte in den Tests an.

Die Antwortklasse AC_{23}: *„Fehler bei Methodenaufruf mit Eingaben, die kein Feld sind, wird abgefangen“* nimmt sie hingegen bewusst nicht auf. In dieser frühen Phase der Veranstaltung liegt der Fokus auf dem sicheren Umgang mit Feldern. Die Berücksichtigung von robustem Fehlermanagement für den Fall, dass kein Feld als Eingabe übergeben wird, würde aus ihrer Sicht die Studierenden inhaltlich überfordern und geht über die intendierten Lernziele hinaus.

Darüber hinaus berücksichtigt Clara Code ein Fehlermuster, das im letzten Jahr besonders häufig bei ähnlichen Übungsaufgaben aufgetreten ist: Viele Studierende haben das Ergebnis nicht wie gefordert mittels `return`-Anweisung zurückgegeben, sondern es lediglich über die Konsole ausgegeben. Um dieses Verhalten systematisch erfassen und im Feedback adressieren zu können, ergänzt sie die folgende Antwortklasse:

AC_{24}: Die Methode `<Methodenname>` hat den Rückgabetyp `void` und gibt das korrekte Ergebnis auf der Konsole aus.

Damit die Diagnose-Komponente von `APFEL` diese Antwortklasse korrekt zuordnen kann, ist eine Kombination aus statischer und dynamischer Analyse notwendig. Statisch lässt sich mittels einer GreQL-Abfrage prüfen, ob die Methode den Rückgabetyp `void` besitzt. Dynamisch werden die relevanten Unit-Tests so angepasst, dass der Standardausgabestream (`System.out`) durch einen ersetzbaren Puffer-Stream substituiert wird. Die Testfälle können dadurch nicht nur erfassen, ob ein Wert ausgegeben wurde, sondern auch, ob dieser dem erwarteten Minimum des Eingabefeldes entspricht. Um den Vergleich robust zu gestalten, ersetzt Clara Code den Standard-`OutputStream` durch eine eigene Testinstanz, die die Konsolenausgaben abfängt. Mithilfe regulärer Ausdrücke werden die im Output enthaltenen Zahlenwerte extrahiert, sodass auch dann eine korrekte Diagnose erfolgen kann, wenn Studierende zusätzlich erläuternden Text ausgeben (z. B. „Das Minimum ist: 3”). Dieser Ansatz ist zwar nicht in der Lage, alle denkbaren Formate von Ausgaben zuverlässig zu verarbeiten, etwa komplexere oder stark abweichend formatierte Strings. Auf Grundlage der Erfahrungen aus dem letzten Semester konnte Clara Code jedoch typische Varianten der Studierendenlösungen identifizieren und mittels regulären Ausdrücken prüfen, ob diese mit bestimmten Mustern übereinstimmen. Dadurch lässt sich die Antwortklasse AC_{24} angemessen zuverlässig diagnostizieren und gezielt für die Generierung von Feedback nutzen.

Die Übersicht in Tabelle 7.1 fasst die verwendeten Antwortklassen und die jeweils eingesetzten Diagnoseverfahren zusammen.

ID	Beschreibung	Diagnoseverfahren
AC_0	Funktional korrekte Implementierung von `minSearch`	Kombination aus AC_1 – AC_3
AC_1	Korrektes Ergebnis für Felder mit positiven Zahlen	Unit-Tests
AC_2	Korrektes Ergebnis für Felder mit negativen Zahlen	Unit-Tests
AC_3	Korrekte Behandlung eines leeren Feldes	Unit-Tests
AC_4	Existenz einer Methode mit dem Bezeichner `minSearch` in `Homework3`	GreQL-Abfrage
AC_{24}	Rückgabetyp `void`, Ergebnis korrekt über Konsole ausgegeben	GreQL-Abfrage (Rückgabetyp) Unit-Tests (Output-Stream)

Tabelle 7.1: Antwortklassen für `minSearch`-Aufgabe und diagnostische Umsetzung

Festlegung von Feedback-Strategien

Im nächsten Schritt hat *Clara Code* die Möglichkeit, individuelle Präferenzen und Feedback-Strategien für die Aufgabe festzulegen. `APFEL` verfügt bereits über ein Set grundlegender Feedback-Strategien, welche sich auf Ergebnisse aus der empirischen Forschung zu Feedback-Wirksamkeit abgeleitet wurden.[2] So haben Studien gezeigt, dass bei **T&E** in Kombination mit **LOGE** (vgl. Kapitel 4) Rückmeldungen vom Typ *Knowledge about Concepts* (`KC`) bei Anfängerinnen und Anfängern am wirksamsten ist. Lernende in dieser Situation profitieren stärker von einem konzeptuellen Hinweis profitieren – etwa einer Erklärung, wie Wiederholungen als Kontrollstruktur korrekt strukturiert werden –, anstatt z. B. lediglich auf einen bereits gemachten Fehler hingewiesen zu werden. Vor kurzem hat Prof. Dr. Clara Code zudem eine Studie gelesen, die nahelegt, dass Feedback-Typen in der Form von Reflexionsfragen bei Fehlern, die auf **SYNE** oder **TYPE** zurückzuführen sind, besonders effektiv sind. In solchen Fällen profitieren Lernende davon, wenn sie durch gezielte Fragen angeregt werden, über die Ursachen ihrer Fehler nachzudenken. Sie wählt daher in `APFEL` aus, dass für diese Kategorien ausschließlich Rückmeldungen vom Typ `REFLECT` generiert werden sollen (siehe Tabelle 4.11).

Darüber hinaus berücksichtigt sie eigene Erfahrungen aus der Lehre: Im vergangenen Semester fiel ihr auf, dass Rückmeldungen des Typs `KM-PI`, die bereits während der Bearbeitung auf mögliche Performance-Probleme hinwiesen, Lernende eher irritierten und ihre aktuelle Bearbeitungs-Strategie unterbrachen. Teilweise führte dies sogar dazu, dass funktionierender Code gelöscht oder nicht weiter bearbeitet wurde. Um dies zu vermeiden, entscheidet Clara Code, dass Feedback vom Typ `KM-PI` nur dann ausgelöst wird, wenn der Interventionspunkt **COMPLETE** diagnostiziert wurde.

7.5.3 Durchlauf eines Beispiel-Studierenden im System

Im Folgenden wird das Zusammenspiel der Systemkomponenten anhand einer fiktiven Studierendenlösung exemplarisch dargestellt. Die Studierende *Ada Loop* bearbeitet die Aufgabe `minSearch` im System `APFEL`.

[2] Die nachfolgenden Befunde sind rein *fiktiv* und dienen ausschließlich zur Illustration.

Ausgangslösung von Ada Loop

Ada schreibt folgenden ersten Lösungsversuch in den Editor:

```
public class Homework3 {
    public void minSearch(int[] arr) {
        int min = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < min) {
                min = arr[i];
            }
        }
        System.out.println("Das Minimum ist: " + min);
    }
}
```

Listing 7.1: Lösungsversuch von Ada zur Aufgabe minSearch

Die Lösung kompiliert fehlerfrei und liefert auf den ersten Blick plausible Ergebnisse. Dennoch enthält sie mehrere typische Probleme:

- Die Methode ist als `void` deklariert und gibt das Ergebnis über die Konsole aus ($\mathbf{AC_{24}}$), anstatt es zurückzugeben.
- Sonderfälle wie ein leeres Feld werden nicht behandelt ($\mathbf{AC_3}$ nicht erfüllt).

Diagnose-Komponente

Die Diagnose-Komponente überprüft Adas Lösung zunächst anhand der hinterlegten Unit-Tests und GreQL-Regeln. Dabei ergibt sich folgender Befund:

- $\mathbf{AC_4}$ erfüllt (Methode `minSearch` existiert).
- $\mathbf{AC_1}$ und $\mathbf{AC_2}$ erfüllt (korrektes Ergebnis für Felder mit positiven/negativen Zahlen).
- $\mathbf{AC_3}$ nicht erfüllt (kein Handling für leere Felder).
- $\mathbf{AC_{24}}$ erfüllt (void-Rückgabe, korrekter Wert über `System.out.println`).

Damit wird der IST-Zustand als Kombination der erfüllten Antwortklassen erfasst und dem definierten SOLL-Zustand gegenübergestellt, der $\mathbf{AC_0}$ (funktional korrekt, mit `return`-Wert) erfordert.

Trigger-Komponente

Auf Basis der Diagnoseergebnisse prüft die Trigger-Komponente, ob ein Interventionspunkt vorliegt.

- Durch die Verletzung von $\mathbf{AC_3}$ wird ein **LOGE**-Fehler (fehlende Sonderfallbehandlung) diagnostiziert.
- Durch die Erfüllung von $\mathbf{AC_{24}}$ wird ein spezifisches Fehlermuster identifiziert (Konsole statt Rückgabewert).

Für beide Situationen greifen hinterlegte Regeln:

- Bei **LOGE**-Fehlern in Kombination mit **T&E**-Verhalten (hier: wiederholte Abgaben mit ähnlicher Struktur) wird laut hypothetischer Befunde (vgl. Abschnitt 7.5.2) Feedback vom Typ *Knowledge about Concepts (KC)* ausgelöst.
- Für $\mathbf{AC_{24}}$ hat Clara Code individuell festgelegt, dass immer direkt Feedback vom Typ *Knowledge about Task Constraints (KTC)* gegeben wird.

Kontextualisierer

Der Kontextualisierer reichert die Diagnoseinformationen mit situativen und instruktionalen Faktoren an:

- Lernziel: Anwendung des Konzepts *loop*.
- Relevante Konzepte: *Feld*, *Minimum*, *Methodenrückgabe*.
- Vorwissen: grundlegende Kontrollelemente (if/else, Schleifen).

Zusätzlich erhählt das System die Information aus dem Lernendenmodell von ALEA, dass Ada die Konzepte `loop` und `array` bereits mit hoher Wahrscheinlichkeit beherrscht. Das Konzept `MinMax?minimum` jedoch noch keinen Wert enthält.

Feedback-Generator

Die Regelentscheidungen und Kontextdaten werden in Prompt-Vorlagen überführt. Für das fehlende Rückgabeverhalten ($\mathbf{AC_{24}}$) wird eine `KTC`-Rückmeldung erzeugt. Das Sprachmodell erhält hierzu u. a. folgende Eingaben: Aufgabenbeschreibung, Musterlösung, Hinweis auf das Konzept *Methodenrückgabe*.

Ein mögliches Feedback könnte lauten:

> *„Ihre Methode `minSearch` berechnet das Minimum korrekt, gibt es aber nur über die Konsole aus. Die Aufgabenstellung verlangt jedoch, dass der Wert als Rückgabewert der Methode zur Verfügung steht. Überlegen Sie, wie Sie Ihre Implementierung anpassen können, sodass `minSearch` einen `int` zurückliefert.“*

Für den fehlenden Sonderfall ($\mathbf{AC_3}$) wird zusätzlich ein KC-Feedback bereitgestellt, z. B.:

> *„Bedenken Sie, dass ein Feld auch leer sein kann. In diesem Fall ist der Zugriff auf das erste Element nicht möglich und führt zu einem Fehler. Überlegen Sie, wie Sie Ihre Methode erweitern können, um diesen Sonderfall abzufangen.“*

Ergebnis

Ada Loop erhält im System beide Rückmeldungen kombiniert: eine konzeptuelle Erinnerung an die Behandlung von Sonderfällen sowie einen expliziten Hinweis auf die falsche Verwendung der Rückgabe. Damit wird sichtbar, wie Diagnose, Trigger, Kontextualisierung und Feedback-Generator in `APFEL` zusammenspielen und regelgeleitet ein inhaltlich konsistentes, kontextsensitives Feedback bereitstellen.

7.6 Diskussion und Limitationen

Mit APFEL liegt ein konzeptueller Systementwurf vor, der die in den vorangegangenen Kapiteln gewonnenen theoretischen und empirischen Einsichten zu Feedback-Strategien, Aufgabenformalismen und kontextuellen Einflussfaktoren in einem kohärenten technischen Rahmen bündelt. Im Sinne des Design Science Research-Prozesses (vgl. Abschnitt 3.3.1) ist APFEL dabei nicht als abgeschlossenes Produkt zu verstehen, sondern als Artefakt im *Rigor Cycle*, dessen Funktion in der Konkretisierung, Überprüfung und Weiterentwicklung zentraler theoretischer Annahmen liegt. Entsprechend ist das System weniger als finales Design, sondern vielmehr als forschungs- und entwicklungspraktisches Framework zu verstehen, das sowohl die empirische Untersuchung adaptiver Feedbackprozesse als auch die prototypische Umsetzung kontextsensitive Lernsysteme unterstützt.

Um die prinzipielle technische Umsetzbarkeit des Systementwurfs zu prüfen, wurde APFEL prototypisch implementiert und in einer ersten Pilotierung in einer Schulklasse eingesetzt (Lohr, Wechsler und Berges 2025). Die folgenden Abbildungen zeigen exemplarisch die Anwendersicht des Prototyps. Sie erheben keinen Anspruch auf Vollständigkeit der im Kapitel beschriebenen Architektur, sondern dienen ausschließlich der Illustration des Artefaktcharakters.

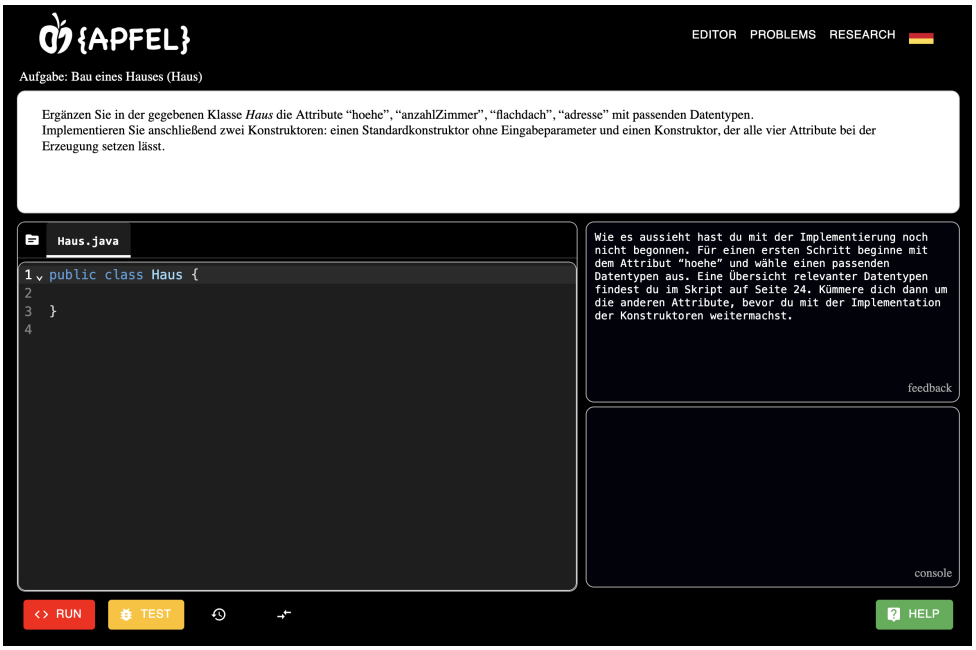


Abbildung 7.5: Benutzeroberfläche des APFEL-Prototyps – Bearbeitung einer Aufgabe

Abbildung 7.5 zeigt die allgemeine Benutzeroberfläche des Prototyps während der Bearbeitung einer Aufgabe. In diesem Szenario stand den Lernenden die Möglichkeit offen, das eigene Programm auszuführen (RUN), gezielt Testläufe zu starten (TEST) oder Unterstützung anzufordern (HELP). Auf diese Weise konnten Lernende den Bearbeitungsprozess selbst steuern und bei Bedarf verschiedene Formen von Rückmeldung bzw. Feedback anfordern.

Abbildung 7.6 zeigt die Darstellung, die nach der Ausführung von Tests erscheint. Hier werden die für die jeweilige Aufgabe hinterlegten Antwortklassen (bzw. wahlweise die zugehörigen Testfälle) angezeigt. Lernende erhalten so unmittelbare Rückmeldung darüber, welche funktionalen und strukturellen Anforderungen ihre Lösung bereits erfüllt und an welchen Stellen noch Abweichungen bestehen. Damit werden die im System verankerten diagnostischen Mechanismen sichtbar, die den IST-Zustand einer Lösung bestimmen.

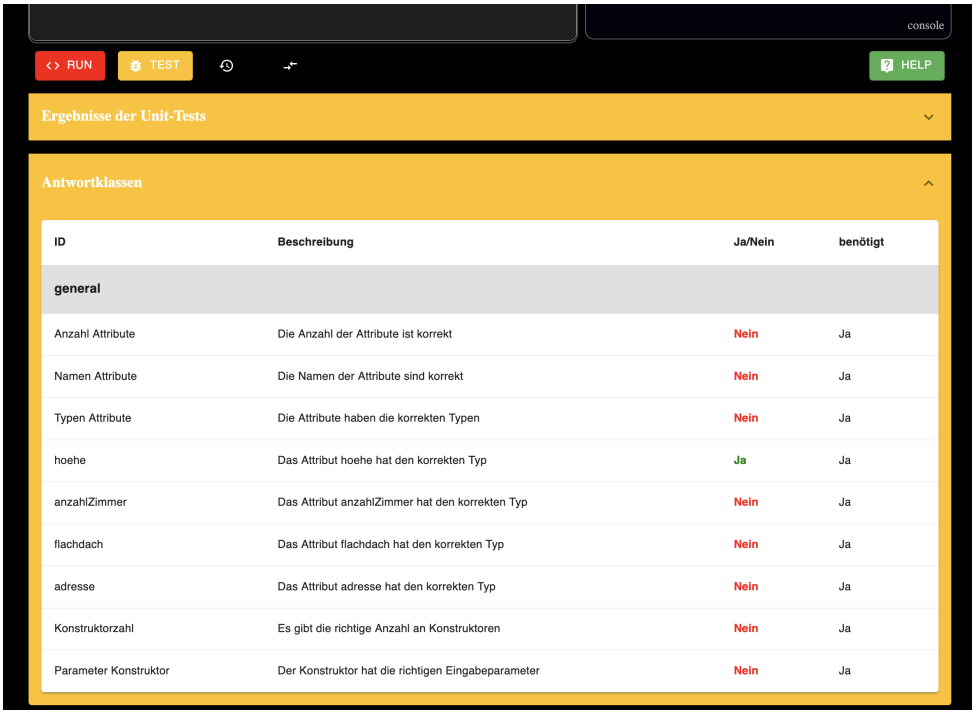


Abbildung 7.6: Übersicht der (erfüllten) Antwortklassen im APFEL-Prototyp

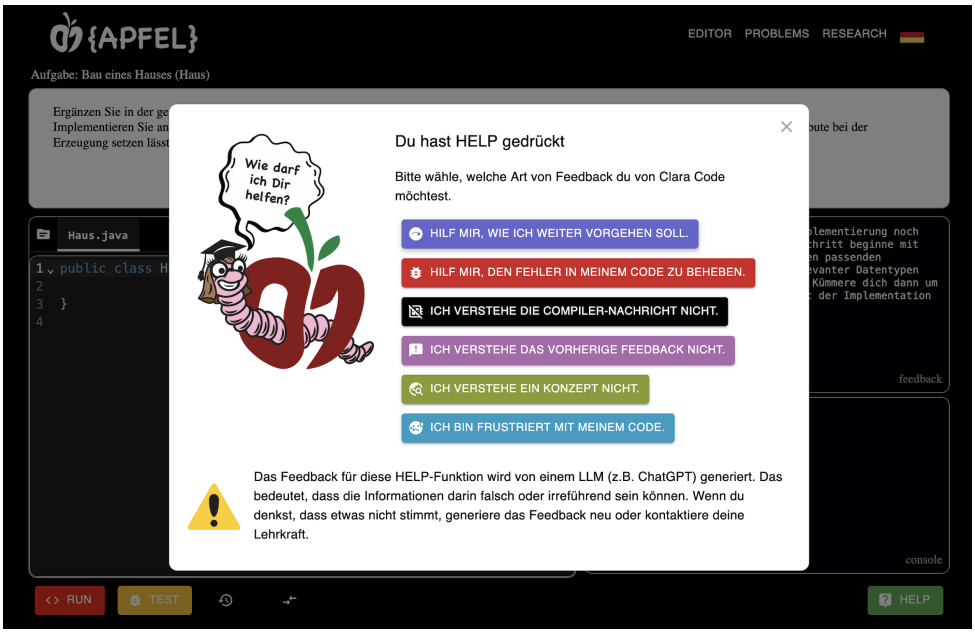


Abbildung 7.7: Anforderung und Auswahl von Feedback-Typen im APFEL-Prototyp

Abbildung 7.7 zeigt den Zustand der Benutzeroberfläche, wenn Lernende aktiv über die HELP-Funktion Unterstützung anfordern. In dieser Pilotierung standen verschiedene Typen von Feedback zur Auswahl, die jeweils unterschiedlichen didaktischen Funktionen entsprachen. Nach der Auswahl wird der entsprechende Feedback-Typ automatisch generiert und im Feedback-Fenster angezeigt.

Erste Pilotversuche in schulischen Kontexten zeigten eine prinzipielle Akzeptanz und positive Wahrnehmung des Systems durch Lernende (Lohr, Wechsler und Berges 2025). Eine systema-

tische Evaluation steht jedoch noch aus. Dies ist nicht primär eine technische, sondern eine forschungslogische Einschränkung: `APFEL` verfolgt das Ziel Feedback-*Strategien* zu ermöglichen, deren evidenzbasierte Spezifikation für viele situations- und aufgabenspezifische Konstellationen derzeit noch fehlt. Solange robuste, empirisch abgesicherte Strategierepertoires begrenzt sind, lässt sich die Systemwirksamkeit nur eingeschränkt kausal zuschreiben. In dieser Phase fungiert `APFEL` daher vor allem als Forschungsinfrastruktur: Die explizite Trennung der Systemkomponenten erlaubt kontrollierte Studien (A/B-Vergleiche von Strategien, Regelkonfigurationen und Feedback-Typen), den direkten Vergleich regelbasierter und LLM-gestützter Generierung sowie Hybridansätze. Darüber hinaus lassen sich Hypothesen der Feedbackforschung in kontrollierte technische Experimente überführen – etwa ob konzeptuelles Feedback in bestimmten Aufgabenphasen effektiver ist als prozedurales oder inwiefern *situativ adaptierte Frequenz und Timing* von Rückmeldungen (z. B. Auslösen nach diagnostizierten Stagnationsmustern) die Selbstregulation fördern. Die Ergebnisse solcher Untersuchungen können iterativ in die Regelbasis und die Prompt-Konfigurationen zurückfließen und so den Artefaktcharakter im Sinne eines DSR-Zyklus stärken.

Ein zentrales Leitmotiv von `APFEL` ist dabei die Transparenz der Feedbackentscheidungen. In der Lehrpraxis ist Nachvollziehbarkeit von Feedbackprozessen nicht nur aus didaktischer, sondern auch aus ethischer Perspektive wesentlich: Lernende und Lehrende sollten nachvollziehen können, auf welcher Grundlage ein System eine bestimmte Rückmeldung generiert. Transparente Entscheidungsprozesse fördern Vertrauen und Verständnis und wirken der Wahrnehmung „black-box"-artiger Systeme entgegen (Chaudhry, Cukurova und Luckin 2022). Die Architektur von `APFEL` trägt diesem Anspruch Rechnung, indem alle Entscheidungen auf expliziten, beobachtbaren Evidenzen (Antwortklassen, Kontextdaten) beruhen und dokumentiert werden können.

Für die Forschung eröffnet dieser Ansatz die Möglichkeit, Feedbackprozesse nicht nur inhaltlich, sondern auch strukturell zu analysieren: Da sich jede Rückmeldung in `APFEL` auf die zugrunde liegenden Diagnose-, Kontext- und Regelentscheidungen zurückführen lässt, können sowohl technische als auch didaktische Hypothesen überprüft werden – etwa ob bestimmte Entscheidungsregeln in bestimmten Lernkontexten zu förderlicheren Lernverläufen führen.

Die Diagnose-Komponente folgt der Logik der Antwortklassen und stützt sich damit auf objektivierbare Kriterien. Sie erfasst ausschließlich Merkmale, die beobachtbar und regelgeleitet überprüfbar sind – etwa funktionale Korrektheit, syntaktische Struktur oder charakteristische Fehlermuster. Ihre Stärke liegt in der systematischen und nachvollziehbaren Abbildung solcher Evidenzen. Zugleich stößt dieser Ansatz dort an Grenzen, wo Programmierleistungen qualitative, stilistische oder ästhetische Dimensionen annehmen, die sich nicht eindeutig operationalisieren lassen. Kriterien wie Lesbarkeit, Eleganz, Effizienz oder Verständlichkeit von Code unterliegen meist einem interpretativen Urteil und sind damit mit dem vorliegenden Ansatz nur eingeschränkt formal beschreibbar. Zwar können Heuristiken oder Style-Guides als Näherungen dienen, doch bleiben solche Aspekte kontext- und subjektabhängig und entziehen sich letztlich einer objektiven, regelgeleiteten Diagnose im Sinne der Antwortklassenlogik.

Über die genannten Grenzen hinaus stößt die Diagnose-Komponente auch dort an ihre Grenzen, wo Rückmeldungen auf tieferliegende konzeptuelle Misskonzepte abzielen. Wie bereits Draper (2005) hervorhebt, lassen sich solche Fehlvorstellungen kaum durch regelbasierte Verfahren adressieren, da sie auf falschen mentalen Verknüpfungen zwischen Konzepten beruhen und sich somit nicht unmittelbar in beobachtbaren Artefakten des Programmcodes widerspiegeln. Um diese zu erkennen und gezielt zu korrigieren, ist eine Modellierung der Lernenden erforderlich, die kognitive Zustände und Wissensbeziehungen erfasst. Die in `APFEL` vorgesehene Kopplung mit einem Lernendenmodell ist daher ein entscheidender nächster Schritt, um auch komplexe, konzeptuelle

Fehlkonzepte differenziert zu diagnostizieren und langfristig personalisiert zu adressieren. Gleichzeitig sollte jedoch kritisch reflektiert werden, dass Lernendenmodelle immer eine datenbasierte Repräsentation von Personen darstellen und damit Fragen nach Autonomie, Repräsentationsgerechtigkeit und Datenschutz aufwerfen. Wie Weich et al. (2021) argumentiert, erzeugen solche Modelle lediglich „Datendoubles", die Lernende zu statistischen Projektionen ihrer Leistungen verdichten. Eine verantwortungsvolle Systemgestaltung muss daher Mechanismen vorsehen, die die interpretative Offenheit dieser Modelle transparent machen und Fehlrepräsentationen vermeiden.

`APFEL` wurde primär für Einführungsszenarien des Programmierlernens konzipiert, in denen Aufgaben klar strukturierte Eingabe-Ausgabe-Relationen aufweisen. Die Übertragbarkeit auf komplexere Aufgabenformate wie Debugging-, Design- oder Refactoring-Aufgaben erfordert Erweiterungen der Diagnose- und Kontextualisierungskomponenten. In solchen Szenarien können beispielsweise qualitative Kriterien (Effizienz, Lesbarkeit, Stil) oder multiple Lösungswege eine größere Rolle spielen, was eine stärkere Integration semantischer Analysen oder lernendenzentrierter Bewertungslogiken notwendig macht.

Darüber hinaus setzt die Nutzung solcher Lernsysteme ein gewisses Maß an selbstreguliertem Lernen voraus (Ditton 2014). Lernende müssen in der Lage sein, bereitgestelltes Feedback eigenständig zu interpretieren, in ihren Bearbeitungsprozess zu integrieren und daraus geeignete Handlungsstrategien abzuleiten. Ohne grundlegende Fähigkeiten zur Selbststeuerung und Reflexion kann der adaptive Mechanismus von `APFEL` seine intendierte Wirkung womöglich nur eingeschränkt entfalten. In betreuten Lernkontexten kann `APFEL` als assistives Werkzeug fungieren, das Lehrpersonen diagnostische Einblicke liefert und Vorschläge für individualisierte Rückmeldungen bereitstellt. `APFEL` sollte somit nicht als Systementwurf für vollständig autonome Settings verstanden werden, die Lehre ersetzen können. Feedback ist immer in einen sozialen und didaktischen Kontext eingebettet und erfordert pädagogische Verantwortung in seiner Interpretation und Anwendung. Ein rein automatisiertes System liefe Gefahr, Lernende zu über- oder unterfordert und damit zentrale Lernprozesse wie produktives Ringen mit Problemen oder metakognitive Selbstreflexion zu unterminieren. Zudem bleibt die Kontextsensitivität menschlicher Rückmeldungen – etwa im Hinblick auf motivationale, soziale oder emotionale Faktoren – algorithmisch nur begrenzt erfassbar. Auch die normative Verantwortung für Bewertungen und Lernentscheidungen liegt weiterhin bei Lehrpersonen. `APFEL` ist daher als komplementäres, nicht substituierendes Werkzeug zu verstehen, das menschliche Lehrhandlungen unterstützt, Transparenz erhöht und empirisch fundierte Feedback-Strategien skalierbar macht.

Kapitel 8

Empfehlungen und Fazit

Die vorangehenden Kapitel haben aus drei Perspektiven auf dasselbe Problem geblickt: *Wie können Feedback-Strategien modelliert und formalisiert werden, damit kontextsensitives Feedback in Programmierlernsystemen nachvollziehbar, skalierbar und systematisch erforschbar umgesetzt werden kann?* Kapitel 4 rekonstruierte auf empirischer Basis, *wann* Lehrpersonen eingreifen und *welche* Formen von Rückmeldungen sie in typischen Unterrichtssituationen wählen. Kapitel 5 überführte diese Einsichten in ein formales Vokabular aus Antwortklassen und Prozessindikatoren und verortete sie im Y-Modell als konzeptuellem Bezugsrahmen. Kapitel 6 und Kapitel 7 zeigten schließlich, wie diese Strukturen technisch nutzbar werden, insbesondere im Zusammenspiel mit generativen Verfahren.

Vor diesem Hintergrund bündelt das vorliegende Kapitel resultierende *Handlungsempfehlungen* für Wissenschaft und Praxis und zieht ein *Fazit*. Zunächst werden in Abschnitt 8.1 konkrete Implikationen für Systementwicklung, Forschung und Bildungspraxis formuliert, die direkt aus den Ergebnissen der vorangehenden Kapitel ableitbar sind. Daran anschließend fasst Abschnitt 8.2 die zentralen Beiträge der Arbeit zusammen, diskutiert ihre Reichweite und Grenzen und skizziert eine Perspektive für die Weiterentwicklung adaptiver Feedbacksysteme. Beide Teile sind komplementär zu lesen: Während die Empfehlungen auf Umsetzbarkeit und nächste Schritte zielen, verdichtet das Fazit die übergreifende Logik der Arbeit und markiert den theoretisch-konzeptionellen Kern.

8.1 Empfehlungen für Forschung und Praxis

Die in dieser Arbeit entwickelten theoretischen, empirischen und technologischen Beiträge erlauben es, konkrete Empfehlungen für Systementwicklung, Forschung und Unterrichtspraxis abzuleiten. Aufbauend auf den Ergebnissen der Kapitel 4, Abschnitt 5.2, Kapitel 6 und Kapitel 7 werden im Folgenden zentrale Implikationen zusammengefasst, die Anreize für zukünftige Arbeiten und Entwicklungen geben können.

8.1.1 Empfehlungen für die Entwicklung adaptiver Lernsysteme

Für die Gestaltung zukünftiger Lernumgebungen ergibt sich aus den Ergebnissen dieser Dissertation die zentrale Empfehlung, die *Feedbacklogik* von der konkreten *Feedbackgenerierung* zu trennen. Der in Kapitel 7 vorgestellte Systementwurf `APFEL` illustriert einen möglichen Weg, wie adaptive Systeme so aufgebaut werden können, dass die Modellierung und Kontextualisierung von Rückmeldungen regelbasiert, nachvollziehbar und didaktisch steuerbar erfolgt, während die Erzeugung konkreter Rückmeldungen technologieoffen bleibt. Damit wird kein endgültiges Ar-

chitekturmodell festgeschrieben, sondern ein Orientierungsrahmen skizziert, der zeigt, wie sich didaktische Steuerbarkeit und technologische Flexibilität verbinden lassen.

Ein zentrales Entwicklungsprinzip ist dabei die **Kontextualisierung** von Feedback. Wie insbesondere die Analysen in Kapitel 6 verdeutlichen, steigt die Qualität automatisch erzeugter Rückmeldungen, wenn Aufgaben- und Antwortmerkmale sowie relevante Kontextinformationen strukturiert bereitgestellt werden. Systeme sollten daher Mechanismen vorsehen, die solche Kontexte integrieren – etwa durch die semantische Verknüpfung von Aufgaben, erwarteten Antwortmustern (Antwortklassen) und zugehörigen Feedbackausgestaltungen.

Für Umgebungen, die Modelle zur Feedbackgenerierung einsetzen, sollte perspektivisch geprüft werden, inwieweit **domänenspezifische oder feinjustierte Modellvarianten** die Qualität und Passung von Rückmeldungen weiter verbessern können. Die in dieser Arbeit durchgeführten Analysen haben gezeigt, dass bereits Modelle, die auf breiten, allgemeinsprachlichen Datensätzen trainiert wurden, in der Lage sind, didaktisch anschlussfähige Rückmeldungen zu erzeugen (Kapitel 6). Es ist daher naheliegend anzunehmen, dass Modelle, die zusätzlich auf qualitätsgesicherten Lehr-/Lernmaterialien trainiert oder feinjustiert werden, noch gezielter auf fachspezifische Lernkontexte reagieren könnten. Ob und in welchem Umfang sich dadurch tatsächlich eine Qualitätssteigerung erzielen lässt, bleibt Gegenstand zukünftiger empirischer Untersuchungen. Verfahren wie *Retrieval-Augmented Generation* (RAG) erscheinen in diesem Zusammenhang als vielversprechende Ergänzung, da sie es ermöglichen, kuratierte Materialien gezielt einzubinden und so den Kontext für die Feedbackgenerierung weiter anzureichern.

Ein weiteres zentrales Entwicklungsziel ist **Transparenz** gegenüber Lehrenden und Lernenden. Adaptive Systeme sollten nachvollziehbar machen, auf welcher Grundlage Rückmeldungen erzeugt wurden (z. B. genutzte Regeln, herangezogene Kontexte, Evidenzen). Nur so kann Vertrauen in Ergebnisse und die Nutzung entstehen. Das oft vorgebrachte Argument, auch menschliche Lehrpersonen machten Fehler, relativiert die Notwendigkeit von Transparenz dabei nicht: Während Menschen in der Regel Fehler metakognitiv reflektieren und ihr Handeln adaptieren können, verfügen heutige Systeme nicht über ein entsprechendes Selbstüberwachungsvermögen. Umso wichtiger sind technische und didaktische Vorkehrungen zur Fehlerminderung, klare Verantwortlichkeiten und überprüfbare Begründungen.

8.1.2 Empfehlungen für zukünftige Forschung

Die empirische Untersuchung in Kapitel 4 fokussierte auf Feedbackentscheidungen von Lehrpersonen. Zukünftige Arbeiten sollten diese Perspektive erweitern und die Seite der Lernenden systematisch einbeziehen. Es gilt zu erforschen, welche Arten von Rückmeldungen *Lernende* als hilfreich erleben, welche Faktoren – wie Zeitpunkt, Tonalität, Elaborationsgrad – die Nutzung und Akzeptanz von Feedback beeinflussen und welche Indikatoren Lernende selbst als hilfreiche Auslöser für Feedback wahrnehmen: Unter welchen Bedingungen wünschen Lernende Feedback, und inwieweit stimmen diese Feedback-Entscheidungen mit den (rekonstruierten) Begründungskategorien der Lehrenden überein?

Weiterer Forschungsbedarf besteht hinsichtlich der **Wirksamkeit unterschiedlicher Feedback-Strategien**. Die vorliegende Arbeit legt mit einem Kategoriensystem (Kapitel 4) und der formalen Rahmung des Y-Modells eine *Grundlage* für systematische Vergleiche – beschreibt jedoch keine vollständigen Entscheidungslogiken. Künftige Untersuchungen sollten deshalb prüfen, wie sich verschiedene Feedbackformen (z. B. elaborierte, motivationale, strategische Rückmeldungen) auf Lernerfolg, Verständnis und Motivation auswirken und in welchen Kontexten welche Kombinationen besonders effektiv sind.

Ein weiteres Forschungsfeld betrifft die **Diagnostik nicht-ausführbarer Lösungen**. Wie in Abschnitt 5.2 dargestellt, stoßen Verfahren wie Unit-Tests oder statische Codeanalyse bei vielen Antwortklassen an ihre Grenzen. Zukünftige Arbeiten sollten daher untersuchen, wie Ansätze der statischen und dynamischen Codeanalyse, des Natural-Language-Processings sowie formaler Verifikationsmethoden kombiniert werden können, um auch unvollständige oder fehlerhafte Programmierlösungen differenziert zu diagnostizieren – unabhängig davon, ob diese ausführbar sind oder nicht. Darüber hinaus besteht Forschungsbedarf hinsichtlich bislang schwer objektivierbarer Antwortklassen, etwa bei stilistischen Merkmalen wie sprechenden Bezeichnern, Strukturierung oder Kommentierung. Hier stellt sich die Frage, wie solche qualitativen Aspekte in operationalisierbare und beobachtbare Kriterien überführt werden können, um eine möglichst objektive und reproduzierbare Diagnose zu ermöglichen.

Zudem zeigen die Ergebnisse aus Kapitel 6, dass **metakognitive und motivationale Elemente** in automatisierten Rückmeldungen selten auftreten. Es stellt sich die Frage, wie solche Elemente gezielt aktiviert werden können — etwa durch erweiterte Promptstrukturen, eigenständige Module für Reflexionsanreize oder hybride Arrangements, in denen maschinelle Vorschläge durch menschliche Anschlusskommunikation ergänzt werden.

Übergreifend ist die Erforschung von **Fehlerursachen** anstelle bloßer Fehlerfolgen zentral. Feedback sollte nicht nur Symptome (z. B. fehlerhaften Code) adressieren, sondern tieferliegende Ursachen wie Misskonzepte oder unangemessene Strategien. Perspektivisch bieten sich adaptive Diagnosesequenzen und gezielte Folgeaufgaben an, um Fehlvorstellungen sichtbar zu machen und zu prüfen, inwiefern Interventionen ursachenorientiert wirken.

Schließlich ergeben sich auch aus der aktuellen Publikationspraxis methodische Herausforderungen für die Bildungsforschung. Wie Kiesler und Schiffner (2023b) betonen, wird die Reproduzierbarkeit und Weiterverwendung von Forschungsergebnissen in der Computer-Science-Education-Community vor allem durch den eingeschränkten Zugang zu Forschungsdaten behindert. Um die Anschlussfähigkeit und Vergleichbarkeit künftiger Studien zu sichern, sollte daher eine stärkere Orientierung an den Prinzipien offener Wissenschaft – insbesondere die Bereitstellung **offener Datensätze**, **transparenter Annotationen** und **gemeinschaftlich entwickelter Standards** als notwendige Voraussetzung hervorgehoben werden, um so Sekundäranalysen, Replikationsstudien und die nachhaltige Nutzung von Forschungsdaten zu ermöglichen.

8.1.3 Empfehlungen für Lehrpersonen und Bildungspraxis

Auch für die Bildungspraxis ergeben sich mehrere Implikationen: Erstens bietet das in Kapitel 4 entwickelte Kategoriensystem eine strukturierte Grundlage, um Feedbacksituationen zu analysieren und Interventionen begründet zu planen. Es unterstützt die professionelle Reflexion diagnostischer Prozesse, eignet sich für kollegiale Fallbesprechungen und kann in der Aus- und Weiterbildung eingesetzt werden.

Zweitens empfiehlt sich eine **kooperative Entwicklung diagnostischer Ressourcen**. Das in Abschnitt 5.2 vorgestellte Konzept der Antwortklassen und damit verbundene Diagnosewerkzeuge (z. B. Kombination von GreQL-Regeln und Unit-Tests) liefern ein gemeinsames Vokabular, um Fehler- und Antwortvarianten domänenübergreifend zu beschreiben. Eine offene Infrastruktur, in der solche Komponenten geteilt und weiterentwickelt werden, kann die Qualität und Vergleichbarkeit von Unterrichts- und Forschungsergebnissen erhöhen. Erste Vorbilder finden sich in existierenden Systemen (z. B. `JACK` [1]). Ein öffentlicher Austausch entsprechender Regelwerke sollte gezielt gefördert werden.

[1] `https://wiki.uni-due.de/jack/index.php/GReQL_Regeln`

Drittens ist ein **reflektierter Umgang mit automatisiert erzeugten Rückmeldungen** unabdingbar. Solche Rückmeldungen sollten als Ergänzung, nicht als Ersatz pädagogischer Arbeit verstanden werden. Die Analysen in Kapitel 6 zeigen, dass generierte Rückmeldungen trotz hoher sprachlicher Qualität vereinzelt *irreführende Inhalte* enthalten können. Daher ist eine menschliche Qualitätssicherung unerlässlich. So warnt Broussard (2018) vor einer Anthropomorphisierung technischer Systeme: Lehrpersonen sollten Lernende aktiv dabei unterstützen, die Grenzen solcher Rückmeldungen zu verstehen und bei Bedarf menschliche Unterstützung einzuholen.

8.2 Fazit und Ausblick

Die vorliegende Dissertation hatte das Ziel, ein empirisch begründetes und formal anschlussfähiges Vorgehen zu entwickeln, mit dem Feedback-Strategien zum Programmierenlernen so modelliert und beschrieben werden können, dass kontextsensitives Feedback in Programmierlernsystemen nachvollziehbar, skalierbar und systematisch erforschbar bereitgestellt werden kann. Ausgangspunkt war die Beobachtung, dass existierende Programmierlernsysteme zwar in großer Menge *einfaches, überwiegend korrektives Feedback* bereitstellen, jedoch nur selten jene elaborierten, auf Begründungen, Strategien oder Prozessunterstützung zielenden Feedbacktypen, die in der Literatur als besonders lernwirksam beschrieben werden (Narciss 2007; Shute 2008).

8.2.1 Zentrale Ergebnisse und Synthese

Die Arbeit zeigt, dass sich kontextsensitive Feedbackprozesse in der Programmierausbildung auf einer klaren didaktischen und technischen Grundlage systematisieren lassen. Im Zentrum steht die Idee, Entscheidungs- und Diagnoseprozesse so zu modellieren, dass sie maschinell nachvollziehbar und zugleich didaktisch interpretierbar bleiben.

Die empirischen Analysen in Kapitel 4 zeigen, dass Feedback durch Lehrpersonen nicht vorrangig aus Fehlerkorrekturpraktiken besteht, sondern als situative Entscheidung zwischen Intervention und Zurückhaltung. Lehrpersonen beziehen dabei sowohl produktbezogene als auch prozessbezogene Indikatoren ein. Diese Entscheidungslogik kann in begründbare Kategorien überführt werden und bildet damit eine empirisch abgestützte Referenz dafür, welche Situationen überhaupt feedbackwürdig sind.

Das in Kapitel 5 vorgestellte Y-Modell bildet hierfür die konzeptuelle Basis: Es verknüpft Wissen, kognitive Anforderung, Aufgabenstellung und Antwortdimension in einer gemeinsamen Struktur. Aufbauend darauf dienen *Antwortklassen* als diagnostische Brücke zwischen beobachtbarem Lernendenverhalten und der didaktischen Steuerung von Rückmeldungen. Damit werden Feedbackprozesse formal beschreibbar, ohne ihre pädagogische Bedeutung zu verlieren.

Die Untersuchungen zu LLM-basierten Rückmeldungen in Kapitel 6 zeigen ergänzend, dass große Sprachmodelle prinzipiell in der Lage sind, inhaltlich anschlussfähige und elaborierte Rückmeldungen zu generieren. Zugleich wird deutlich: diese Ausgaben können ohne geeignete Rahmung inkonsistent sein und vereinzelt irreführende Inhalte enthalten. Die Evaluationsbefunde bestätigen, dass die Qualität von LLM-basierten Rückmeldungen deutlich steigt, wenn der Modelloutput durch Regeln und formale Repräsentationen gerahmt wird. Damit wird deutlich: *Kontextualisierung ist ein entscheidender Qualitätshebel*, weil sie generative Verfahren auf eine didaktisch kontrollierte Entscheidungsbasis zurückbindet.

Der konzeptuelle Systementwurf `APFEL` in Kapitel 7 zeigt schließlich, dass sich kontextsensitive Feedbackprozesse technisch abbilden lassen, indem man (1) Aufgaben, Antworten und Feedbacktypen semantisch formalisiert, (2) die Feedbacklogik von der sprachlichen Generierung trennt, und (3) große Sprachmodelle gezielt durch strukturierte Kontextinformationen steuert.

8.2.2 Von LLM-generierten Rückmeldungen zu Feedback

Die Ergebnisse verdeutlichen, dass der Weg von „spontanen" LLM-*Rückmeldungen* hin zu didaktisch fundiertem LLM-*Feedback* noch weit ist. Sprachmodelle können zwar elaborierte Texte generieren, ihnen fehlt jedoch das explizite Wissen darüber, *wann*, *warum* und *in welcher Form* eine Rückmeldung lernförderlich ist. Erst durch die Einbettung in eine regelbasierte, kontextreiche Infrastruktur lassen sich ihre Ausgaben als pädagogisch steuerbare Feedbackhandlungen interpretieren.

Das Y-Modell liefert hierfür eine formale Grundlage, indem es die für den Feedbackzeitpunkt und -typ relevanten Merkmale einer Aufgabe explizit macht. Die empirisch erarbeiteten Kategorien und Entscheidungslogiken liefern die didaktischen Indikatoren, an denen sich die Systemsteuerung orientieren kann. Die technologischen Befunde zeigen, wie LLMs in diese Architektur integriert werden können – als generative Komponenten, deren Qualität und Kohärenz über Regelstrukturen, Tests und Soll-Zustände kontrolliert wird.

Kurz gesagt: Um von LLM-Rückmeldungen zu echtem LLM-Feedback zu gelangen, braucht es nicht „mehr Modell", sondern *mehr Struktur*. Regelbasierte Kontexte, formale Aufgabenrepräsentationen und diagnostisch trennscharfe Kategorien sind die Voraussetzung, um generative Systeme didaktisch zu steuern, anstatt sich von ihnen steuern zu lassen.

8.2.3 Adaptiertes und adaptives Feedback

Die Arbeit macht zudem deutlich, dass sich die Zukunft digitaler Feedbacksysteme entlang zweier Entwicklungsstufen denken lässt. *Adaptiertes Feedback* basiert auf klaren Regeln und Kategorien: Für definierte Situationen wird eine passende Rückmeldung gewählt. Dieses Prinzip schafft Transparenz und Nachvollziehbarkeit und bildet die Grundlage für *adaptives Feedback*, das zusätzlich den individuellen Lernendenkontext berücksichtigt. Adaptivität entsteht erst dann, wenn das System in der Lage ist, Lernendenmodelle in Echtzeit zu aktualisieren und Regelwissen mit datenbasierten Signalen zu kombinieren.

Damit erweist sich der im Rahmen der Arbeit vorgestellte Systementwurf `APFEL` als Brücke zwischen beiden Ebenen: Es definiert, welche didaktischen Regeln für welche Situationen gelten (adaptiertes Feedback) und schafft zugleich die technische Voraussetzung, diese Regeln dynamisch an Lernendenkontexte zu koppeln (adaptives Feedback). Ohne diese zweistufige Logik bliebe personalisiertes Feedback entweder zufällig oder intransparent.

8.2.4 Das Henne-Ei-Problem LLM-generierter Rückmeldungen

So deutlich die Potenziale generativer Systeme sind, so klar treten auch ihre Grenzen hervor. Ein zentrales Dilemma besteht darin, dass Lernende genau dann, wenn sie auf Feedback angewiesen sind, in der Regel nicht über die Kompetenzen verfügen, um die Korrektheit und Eignung generierter Rückmeldungen kritisch zu beurteilen. Die Fähigkeit, Feedback zu evaluieren, entsteht erst durch denselben Lernprozess, den das Feedback unterstützen soll. Damit entsteht eine „Henne-Ei-Problematik": Je stärker Lernende auf Unterstützung angewiesen sind, desto weniger können sie deren Qualität verifizieren.

Hier liegt der fundamentale Unterschied zu menschlichem Feedback. Lehrpersonen tragen fachliche Verantwortung und genießen Vertrauen – Sprachmodelle hingegen operieren probabilistisch und bleiben in ihrer Faktentreue prinzipiell unsicher. LLM-generiertes Feedback kann daher nicht als Ersatz für pädagogisch verantwortetes Feedback verstanden werden, sondern nur als dessen *Erweiterung*. Erst eine didaktische Einbettung, gekoppelt mit Verfahren der Qualitätssicherung, kann verhindern, dass Lernende durch fehlerhafte oder irreführende Rückmeldungen fehlgeleitet werden.

8.2.5 Implikationen und Ausblick

Aus den Ergebnissen ergeben sich vier zentrale Prinzipien für die Gestaltung zukünftiger Feedback-Systeme:

1. **Kontext zuerst:** Aufgaben-, Antwort- und Lernkontext müssen strukturiert bereitgestellt werden. Kontextualisierung ist der entscheidende Wirkfaktor, um Relevanz, Präzision und Nachvollziehbarkeit von Feedback zu gewährleisten.
2. **Trennung von Logik und Generierung:** Die didaktische Feedbacklogik sollte regelbasiert und transparent gestaltet werden, während die sprachliche Realisierung technologieoffen erfolgen kann. Diese Entkopplung schafft sowohl pädagogische Nachvollziehbarkeit als auch technische Flexibilität und ermöglicht hybride Architekturen.
3. **Antwortklassen als gemeinsame Sprache:** Antwortklassen fungieren als semantisches Bindeglied zwischen Aufgaben, Lernendenantworten und Feedback. Sie erlauben nicht nur die formale Beschreibung von Soll-Zuständen, sondern auch die Klassifikation tatsächlicher Lernendenprodukte, die Diagnose typischer Fehlermuster und die systematische Verknüpfung mit geeigneten Rückmeldungsstrategien. Damit bilden sie das Vokabular, in dem didaktische Intention, diagnostische Evidenz und technologische Umsetzung zusammengeführt werden.
4. **Operationalisierung von Prozessindikatoren:** Beobachtbare Auslöser wie `T&E`, `PROGR` oder `STEPBACK` (siehe Tabelle 4.11) müssen präzise definiert werden, um Lernprozesse nachvollziehbar zu modellieren. Interpretative Kategorien – etwa `TRUST` oder `STATEOM` – sollten nur eingesetzt werden, wenn sie auf verlässlichen, empirisch validierten Signalen beruhen.

Auf dieser Grundlage lassen sich konkrete Entwicklungsschritte ableiten: (1) die Durchführung von Wirksamkeitsstudien im Feld, um Effekte unterschiedlicher Feedback-Strategien und Regelkonfigurationen empirisch zu überprüfen, (2) die Erweiterung auf komplexere Programmieraufgaben, insbesondere auf nicht-ausführbare oder konzeptionelle Lösungen, die traditionelle Testverfahren überfordern, (3) der Aufbau eines offenen Ressourcenökosystems mit gemeinsam nutzbaren Antwortklassen, Regelwerken und Kontextbeschreibungen, sowie (4) die Weiterentwicklung der Referenzarchitektur `APFEL` zu einer stabilen und interoperablen Infrastruktur für adaptive Feedbackprozesse.

Die zentrale Erkenntnis dieser Arbeit lässt sich in einer prägnanten Formel zusammenfassen: Kontextsensitive Feedbackprozesse werden dann skalierbar, wenn Diagnose, Interpretation und Steuerung formalisiert, Antwortklassen als semantische und diagnostische Anker etabliert und Generierungstechnologien an evidenzbasierte, didaktische Logiken rückgebunden werden.

Damit entsteht ein neues Paradigma für die Programmierausbildung: nicht *mehr Automatisierung* um ihrer selbst willen, sondern der Aufbau einer *didaktisch fundierten, kontextreichen Infrastruktur*, die menschliche Expertise, formale Modelle und generative Technologien produktiv miteinander verbindet. Große Sprachmodelle können in diesem Rahmen leistungsfähige Werkzeuge sein – doch erst, wenn sie innerhalb klar definierter didaktischer Strukturen agieren, werden sie zu einem echten Beitrag für lernwirksames, transparentes und vertrauenswürdiges Feedback.

Literatur

Ala-Mutka, Kirsti M (Juni 2005). „A Survey of Automated Assessment Approaches for Programming Assignments". In: *Computer Science Education* 15.2, S. 83–102. ISSN: 0899-3408, 1744-5175. DOI: 10.1080/08993400500150747.

Aldriye, Hussam, Asma Alkhalaf und Muath Alkhalaf (2019). „Automated Grading Systems for Programming Assignments: A Literature Review". In: *International Journal of Advanced Computer Science and Applications* 10.3. ISSN: 21565570, 2158107X. DOI: 10.14569/IJACSA.2019.0100328.

Aleven, Vincent et al. (Jan. 2006). „Toward Meta-Cognitive Tutoring: A Model of Help Seeking with a Cognitive Tutor." In: *I. J. Artificial Intelligence in Education* 16, S. 101–128.

Aleven, Vincent et al. (2017). „Instruction Based on Adaptive Learning Technologies". In: *Handbook of Research on Learning and Instruction*. Hrsg. von Richard E. Mayer und Patricia A. Alexander. 2. Aufl. New York: Routledge, S. 522–560.

Alzubaidi, Laith et al. (März 2021). „Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions". In: *Journal of Big Data* 8.1, S. 53. ISSN: 2196-1115. DOI: 10.1186/s40537-021-00444-8.

Anderson, John et al. (Apr. 1995). „Cognitive Tutors: Lessons Learned". In: *Journal of the Learning Sciences* 4.2, S. 167–207. ISSN: 1050-8406, 1532-7809. DOI: 10.1207/s15327809jls0402_2.

Anderson, John R. (2005). *Cognitive Psychology and Its Implications*. 6. ed. New York, NY: Worth Publishers. ISBN: 978-0-7167-0110-1.

Anderson, John R. et al. (Feb. 1990). „Cognitive Modeling and Intelligent Tutoring". In: *Artificial Intelligence* 42.1, S. 7–49. ISSN: 00043702. DOI: 10.1016/0004-3702(90)90093-F.

Anderson, Lorin und David Krathwohl, Hrsg. (2001). *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Complete ed. New York: Longman. ISBN: 978-0-321-08405-7 978-0-8013-1903-7.

Azaiz, Imen, Oliver Deckarm und Sven Strickroth (Dez. 2023). „AI-Enhanced Auto-Correction of Programming Exercises: How Effective Is GPT-3.5?" In: *International Journal of Engineering Pedagogy (iJEP)* 13.8, S. 67–83. ISSN: 2192-4880. DOI: 10.3991/ijep.v13i8.45621.

Azaiz, Imen, Natalie Kiesler und Sven Strickroth (Juli 2024). „Feedback-Generation for Programming Exercises With GPT-4". In: *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. Milan Italy: ACM, S. 31–37. ISBN: 979-8-4007-0600-4. DOI: 10.1145/3649217.3653594.

Azaiz, Imen, Felippo Konrad und Sven Strickroth (2025). „Small but Competitive – Evaluating DeepSeek-R1 Among Diverse Open LLMs for Formative Programming Feedback". In: DOI: 10.18420/ABP2025_10.

Bajohr, Hannes und Markus Krajewski (März 2024). *Quellcodekritik. Zur Philologie von Algorithmen*. 1. Aufl. August Verlag (Imprint von MSB Matthes & Seitz Berlin Verlagsgesellschaft mbH). ISBN: 978-3-7518-9020-5. DOI: 10.52438/avaa1004.

Baker, Ryan, Albert T. Corbett und Vincent Aleven (2008). „More Accurate Student Modeling through Contextual Estimation of Slip and Guess Probabilities in Bayesian Knowledge Tracing". In: *Intelligent Tutoring Systems*. Hrsg. von David Hutchison et al. Bd. 5091. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 406–415. ISBN: 978-3-540-69130-3 978-3-540-69132-7. DOI: 10.1007/978-3-540-69132-7_44.

Balse, Rishabh et al. (Juni 2023). „Investigating the Potential of GPT-3 in Providing Feedback for Programming Assessments". In: *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. Turku Finland: ACM, S. 292–298. ISBN: 979-8-4007-0138-2. DOI: 10.1145/3587102.3588852.

Bangert-Drowns, Robert L. et al. (Juni 1991). „The Instructional Effect of Feedback in Test-Like Events". In: *Review of Educational Research* 61.2, S. 213–238. ISSN: 0034-6543, 1935-1046. DOI: 10.3102/00346543061002213.

Baron, Aviad und Dror G. Feitelson (Juli 2024). „Why Is Recursion Hard to Comprehend? An Experiment with Experienced Programmers in Python". In: *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. Milan Italy: ACM, S. 115–121. ISBN: 979-8-4007-0600-4. DOI: 10.1145/3649217.3653636.

Bauer, Elisabeth et al. (Aug. 2025). „Personalizing Simulation-Based Learning in Higher Education". In: *Learning and Individual Differences* 122, S. 102746. ISSN: 10416080. DOI: 10.1016/j.lindif.2025.102746.

Bayerlein, Oliver (Dez. 2010). „Lernerbeobachtungen Zur Nutzung von Feedback Bei Einem Videogestützten Online-Sprachkurs Für Deutsch Als Fremdsprache". In: *Informationen Deutsch als Fremdsprache* 37.6, S. 570–576. ISSN: 2511-0853, 0724-9616. DOI: 10.1515/infodaf-2010-0605.

Baz, Mehmet Ali und Sevil Hasırcı Aksoy (Aug. 2025). „The Effect of Feedback on Informative Text Writing: AI or Teacher?" In: *Open Praxis* 17.3. ISSN: 2304-070X, 1369-9997. DOI: 10.55982/openpraxis.17.3.871.

Becker, Brett A. et al. (März 2023). „Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. Toronto ON Canada: ACM, S. 500–506. ISBN: 978-1-4503-9431-4. DOI: 10.1145/3545945.3569759.

Becker, Brett A. et al. (2024). „Generative AI in Introductory Programming". In: *Computer Science Curricula 2023*. Hrsg. von Amruth N. Kumar et al. New York, NY, USA: Association for Computing Machinery, S. 438–439. ISBN: 979-8-4007-1033-9.

Bengtsson, Douglas und Axel Kaliff (2023). *Assessment Accuracy of a Large Language Model on Programming Assignments*.

Benjamin, Ludy T. (Sep. 1988). „A History of Teaching Machines." In: *American Psychologist* 43.9, S. 703–712. ISSN: 1935-990X, 0003-066X. DOI: 10.1037/0003-066X.43.9.703.

Bennedsen, Jens und Michael E. Caspersen (Apr. 2019). „Failure Rates in Introductory Programming: 12 Years Later". In: *ACM Inroads* 10.2, S. 30–36. ISSN: 2153-2184, 2153-2192. DOI: 10.1145/3324888.

Berges, Marc, Andreas Mühling und Peter Hubwieser (Nov. 2012). „The Gap between Knowledge and Ability". In: *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*. Koli Finland: ACM, S. 126–134. ISBN: 978-1-4503-1795-5. DOI: 10.1145/2401796.2401812.

Bernius, Jan Philip, Stephan Krusche und Bernd Bruegge (2022). „Machine Learning Based Feedback on Textual Student Answers in Large Courses". In: *Computers and Education: Artificial Intelligence* 3, S. 100081. ISSN: 2666920X. DOI: 10.1016/j.caeai.2022.100081.

Betzendahl, Jonas, Michael Kohlhase und Dennis Müller (2024). „Guided Tours in ALeA: Assembling Tailored Educational Dialogues from Semantically Annotated Learning Objects". In: *Artificial Intelligence. ECAI 2023 International Workshops*. Hrsg. von Sławomir Nowaczyk et al. Bd. 1948. Cham: Springer Nature Switzerland, S. 397–408. ISBN: 978-3-031-50484-6 978-3-031-50485-3. DOI: 10.1007/978-3-031-50485-3_39.

Black, Paul und Dylan Wiliam (März 1998). „Assessment and Classroom Learning". In: *Assessment in Education: Principles, Policy & Practice* 5.1, S. 7–74. ISSN: 0969-594X, 1465-329X. DOI: 10.1080/0969595980050102.

Bloom, Benjamin (Juni 1984). „The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring". In: *Educational Researcher* 13.6, S. 4–16. ISSN: 0013-189X, 1935-102X. DOI: 10.3102/0013189X013006004.

Bloom, Benjamin (1986). *Taxonomy of Educational Objectives. 1: Cognitive Domain*. 29. print. London: Longman. ISBN: 978-0-582-28010-6.

Boguslawski, Samuel, Rowan Deer und Mark G. Dawson (Feb. 2025). „Programming Education and Learner Motivation in the Age of Generative AI: Student and Educator Perspectives". In: *Information and Learning Sciences* 126.1/2, S. 91–109. ISSN: 2398-5348, 2398-5356. DOI: 10.1108/ILS-10-2023-0163.

Boud, David und Elizabeth Molloy, Hrsg. (2013). *Feedback in Higher and Professional Education: Understanding It and Doing It Well*. London ; New York: Routledge. ISBN: 978-0-415-69228-1 978-0-415-69229-8 978-0-203-07433-6.

Brackbill, Yvonne et al. (Juli 1963). „Amplitude of Response and the Delay-Retention Effect." In: *Journal of Experimental Psychology* 66.1, S. 57–64. ISSN: 0022-1015. DOI: 10.1037/h0043368.

Bracken, David W., Carol W. Timmreck und Allan H. Church, Hrsg. (2001). *The Handbook of Multisource Feedback*. 1., Auflage. New York, NY: John Wiley & Sons. ISBN: 978-0-7879-5286-0 978-0-7879-5856-5.

Brandmo, Christian und Siv M. Gamlem (Mai 2025). „Students' Perceptions and Outcome of Teacher Feedback: A Systematic Review". In: *Frontiers in Education* 10, S. 1572950. ISSN: 2504-284X. DOI: 10.3389/feduc.2025.1572950.

Bransford, John D., Ann L. Brown und Rodney R. Cocking (Aug. 2000). *How People Learn: Brain, Mind, Experience, and School: Expanded Edition*. Washington, D.C.: National Academies Press, S. 9853. ISBN: 978-0-309-07036-2. DOI: 10.17226/9853.

Brocker, Annabell et al. (2024). „Eine Taxonomie zur Variabilisierung von Programmieraufgaben". In: DOI: 10.18420/DELFI2024-WS-12.

Broussard, Meredith (Apr. 2018). *Artificial Unintelligence: How Computers Misunderstand the World*. The MIT Press. ISBN: 978-0-262-34673-3. DOI: 10.7551/mitpress/11022.001.0001.

Bruner, Jerome S. (Apr. 1966). „Toward a Theory of Instruction". In: *The bulletin of the National Association of Secondary School Principals* 50.309, S. 304–312. ISSN: 2471-3317. DOI: 10.1177/019263656605030929.

Bruner, Jerome S. (1996). *The Culture of Education*. Cambridge, Mass: Harvard University Press. ISBN: 978-0-674-17952-3.

Bui, Giang et al. (März 2023). „Prior Programming Experience: A Persistent Performance Gap in CS1 and CS2". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. Toronto ON Canada: ACM, S. 889–895. ISBN: 978-1-4503-9431-4. DOI: 10.1145/3545945.3569752.

Busch, Rolf, Hrsg. (2000). *Mitarbeitergespräch - Führungskräftefeedback: Instrumente in der Praxis*. Forschung und Weiterbildung für die betriebliche Praxis 21. München Mering: Hampp. ISBN: 978-3-87988-481-0.

Butler, Deborah L. und Philip H. Winne (Sep. 1995). „Feedback and Self-Regulated Learning: A Theoretical Synthesis". In: *Review of Educational Research* 65.3, S. 245–281. ISSN: 0034-6543, 1935-1046. DOI: 10.3102/00346543065003245.

Butler, Ruth (Dez. 1987). „Task-Involving and Ego-Involving Properties of Evaluation: Effects of Different Feedback Conditions on Motivational Perceptions, Interest, and Performance." In: *Journal of Educational Psychology* 79.4, S. 474–482. ISSN: 1939-2176, 0022-0663. DOI: 10.1037/0022-0663.79.4.474.

Byrka, Marian F. et al. (2021). „A New Dimension of Learning in Higher Education: Algorithmic Thinking". In: *Propósitos y Representaciones* 9.SPE2. ISSN: 23077999, 23104635. DOI: 10.20511/pyr2021.v9nSPE2.990.

Candel, Carmen et al. (Okt. 2020). „Effects of Timing of Formative Feedback in Computer-assisted Learning Environments". In: *Journal of Computer Assisted Learning* 36.5, S. 718–728. ISSN: 0266-4909, 1365-2729. DOI: 10.1111/jcal.12439.

Chang, Yupeng et al. (Juni 2024). „A Survey on Evaluation of Large Language Models". In: *ACM Transactions on Intelligent Systems and Technology* 15.3, S. 1–45. ISSN: 2157-6904, 2157-6912. DOI: 10.1145/3641289.

Chaudhry, Muhammad Ali, Mutlu Cukurova und Rose Luckin (2022). „A Transparency Index Framework for AI in Education". In: *Artificial Intelligence in Education. Posters and Late Breaking Results, Workshops and Tutorials, Industry and Innovation Tracks, Practitioners' and Doctoral Consortium*. Hrsg. von Maria Mercedes Rodrigo et al. Bd. 13356. Cham: Springer International Publishing, S. 195–198. ISBN: 978-3-031-11646-9 978-3-031-11647-6. DOI: 10.1007/978-3-031-11647-6_33.

Chiang, Wei-Lin et al. (2024). „Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference". In: *Proceedings of the 41st International Conference on Machine Learning*. ICML'24. Vienna, Austria: JMLR.org.

Chou, Chih-Yuan und Wen-Jing Chen (Dez. 2025). „Exploring the Messenger Effect on Consumer Emotions and Attitudes: Promoting Socially Responsible Practices in the Cosmetics Sector". In: *Electronic Markets* 35.1, S. 61. ISSN: 1019-6781, 1422-8890. DOI: 10.1007/s12525-025-00811-w.

Christiano, Paul et al. (2017). *Deep Reinforcement Learning from Human Preferences*. DOI: 10.48550/ARXIV.1706.03741.

Cipriano, Bruno Pereira und Pedro Alves (Juni 2023). „GPT-3 vs Object Oriented Programming Assignments: An Experience Report". In: *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. Turku Finland: ACM, S. 61–67. ISBN: 979-8-4007-0138-2. DOI: 10.1145/3587102.3588814.

Cohen, Jacob (Apr. 1960). „A Coefficient of Agreement for Nominal Scales". In: *Educational and Psychological Measurement* 20.1, S. 37–46. ISSN: 0013-1644, 1552-3888. DOI: 10.1177/001316446002000104.

Cope, Bill und Mary Kalantzis (Nov. 2023). „A Little History of E-Learning: Finding New Ways to Learn in the PLATO Computer Education System, 1959–1976". In: *History of Education* 52.6, S. 905–936. ISSN: 0046-760X, 1464-5130. DOI: 10.1080/0046760X.2022.2141353.

Corbett, Albert und John Anderson (1995). „Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge". In: *User Modelling and User-Adapted Interaction* 4.4, S. 253–278. ISSN: 0924-1868, 1573-1391. DOI: 10.1007/BF01099821.

Corradini, Isabella, Michael Lodi und Enrico Nardelli (2018). „An Investigation of Italian Primary School Teachers' View on Coding and Programming". In: *Informatics in Schools. Fundamentals of Computer Science and Software Engineering*. Hrsg. von Sergei N. Pozdniakov und Valentina Dagienė. Bd. 11169. Cham: Springer International Publishing, S. 228–243. ISBN: 978-3-030-02749-0 978-3-030-02750-6. DOI: 10.1007/978-3-030-02750-6_18.

Coy, Wolfgang (Feb. 2008). „Kulturen – nicht betreten? Anmerkungen zur „Kulturtechnik Informatik"". In: *Informatik-Spektrum* 31.1, S. 30–34. ISSN: 0170-6012, 1432-122X. DOI: 10.1007/s00287-007-0207-z.

Creswell, Antonia, Murray Shanahan und Irina Higgins (2022). *Selection-Inference: Exploiting Large Language Models for Interpretable Logical Reasoning*. DOI: 10.48550/ARXIV.2205.09712.

Crow, Tyne, Andrew Luxton-Reilly und Burkhard Wuensche (Jan. 2018). „Intelligent Tutoring Systems for Programming Education: A Systematic Review". In: *Proceedings of the 20th Australasian Computing Education Conference*. Brisbane Queensland Australia: ACM, S. 53–62. ISBN: 978-1-4503-6340-2. DOI: 10.1145/3160489.3160492.

Culbertson, Michael J. (Jan. 2016). „Bayesian Networks in Educational Assessment: The State of the Field". In: *Applied Psychological Measurement* 40.1, S. 3–21. ISSN: 0146-6216, 1552-3497. DOI: 10.1177/0146621615590401.

D'Mello, Sidney K., Blair Lehman und Natalie Person (Mai 2010). „Expert Tutors Feedback Is Immediate, Direct, and Discriminating." In: *Proceedings of the 23rd International Florida Artificial Intelligence Research Society Conference, FLAIRS-23*.

Dainton, Nora (2018). *Feedback in der Hochschullehre*. utb Didaktik 4891. Stuttgart: UTB. ISBN: 978-3-8252-4891-8.

Dede, Christopher (Apr. 1986). „A Review and Synthesis of Recent Research in Intelligent Computer-Assisted Instruction". In: *International Journal of Man-Machine Studies* 24.4, S. 329–353. ISSN: 00207373. DOI: 10.1016/S0020-7373(86)80050-5.

Denning, Peter J. und Matti Tedre (Mai 2019). *Computational Thinking*. The MIT Press. ISBN: 978-0-262-35341-0. DOI: 10.7551/mitpress/11740.001.0001.

Denny, Paul, Viraj Kumar und Nasser Giacaman (März 2023). „Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. Toronto ON Canada: ACM, S. 1136–1142. ISBN: 978-1-4503-9431-4. DOI: 10.1145/3545945.3569823.

Denny, Paul et al. (Mai 2021). „On Designing Programming Error Messages for Novices: Readability and Its Constituent Factors". In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Yokohama Japan: ACM, S. 1–15. ISBN: 978-1-4503-8096-6. DOI: 10.1145/3411764.3445696.

Devecka, Martin (Dez. 2013). „Did the Greeks Believe in Their Robots?" In: *The Cambridge Classical Journal* 59, S. 52–69. ISSN: 1750-2705, 2047-993X. DOI: 10.1017/S1750270513000079.

Devedzic, Vladan, Dragan Djuric und Dragan Gasevic (2009). *Model Driven Engineering and Ontology Development*. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-00281-6 978-3-642-00282-3. DOI: 10.1007/978-3-642-00282-3.

Dhariwal, Prafulla et al. (2020). *Jukebox: A Generative Model for Music*. DOI: 10.48550/ARXIV.2005.00341.

Ditton, Hartmut, Hrsg. (2014). *Feedback und Rückmeldungen: theoretische Grundlagen, empirische Befunde, praktische Anwendungsfelder*. Münster: Waxmann. ISBN: 978-3-8309-3090-7.

Döller, Victoria, Dimitris Karagiannis und Wilfrid Utz (Feb. 2023). „MetaMorph: Formalization of Domain-Specific Conceptual Modeling Methods—an Evaluative Case Study, Juxtaposition and Empirical Assessment". In: *Software and Systems Modeling* 22.1, S. 75–110. ISSN: 1619-1366, 1619-1374. DOI: 10.1007/s10270-022-01047-4.

Dong, Yihuan et al. (Aug. 2021). „You Really Need Help: Exploring Expert Reasons for Intervention During Block-based Programming Assignments". In: *Proceedings of the 17th ACM Conference on International Computing Education Research*. Virtual Event USA: ACM, S. 334–346. ISBN: 978-1-4503-8326-4. DOI: 10.1145/3446871.3469764.

Draper, Steve (Apr. 2005). *Feedback. A Technical Memo*. URL: https://www.psy.gla.ac.uk/~steve/feedback.html.

Ettles, Andrew, Andrew Luxton-Reilly und Paul Denny (Jan. 2018). „Common Logic Errors Made by Novice Programmers". In: *Proceedings of the 20th Australasian Computing Education Conference*. Brisbane Queensland Australia: ACM, S. 83–89. ISBN: 978-1-4503-6340-2. DOI: 10.1145/3160489.3160493.

Evans, Carol (März 2013). „Making Sense of Assessment Feedback in Higher Education". In: *Review of Educational Research* 83.1, S. 70–120. ISSN: 0034-6543, 1935-1046. DOI: 10.3102/0034654312474350.

Fan, Guangrui et al. (März 2025). „The Impact of AI-assisted Pair Programming on Student Motivation, Programming Anxiety, Collaborative Learning, and Programming Performance: A Comparative Study with Traditional Pair Programming and Individual Approaches". In: *International Journal of STEM Education* 12.1, S. 16. ISSN: 2196-7822. DOI: 10.1186/s40594-025-00537-3.

Fan, Lizhou et al. (Okt. 2024). „A Bibliometric Review of Large Language Models Research from 2017 to 2023". In: *ACM Transactions on Intelligent Systems and Technology* 15.5, S. 1–25. ISSN: 2157-6904, 2157-6912. DOI: 10.1145/3664930.

Farrell, Rob, John Anderson und Brian Reiser (Jan. 1984). „An Interactive Computer-Based Tutor for LISP". In: Proceedings of the National Conference on Artificial Intelligence, S. 106–109.

Fellah, Abdelaziz und Ajay Bandi (Mai 2018). „The Essence of Recursion: Reduction, Delegation, and Visualization". In: 33.5, S. 115–123. ISSN: 1937-4771.

Finnie-Ansley, James et al. (Feb. 2022). „The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming". In: *Proceedings of the 24th Australasian Computing Education Conference*. Virtual Event Australia: ACM, S. 10–19. ISBN: 978-1-4503-9643-1. DOI: 10.1145/3511861.3511863.

Finnie-Ansley, James et al. (Jan. 2023). „My AI Wants to Know If This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises". In: *Proceedings of the 25th Australasian Computing Education Conference*. Melbourne VIC Australia: ACM, S. 97–104. ISBN: 978-1-4503-9941-8. DOI: 10.1145/3576123.3576134.

Fournier-Viger, Philippe et al. (Okt. 2013). „A Multiparadigm Intelligent Tutoring System for Robotic Arm Training". In: *IEEE Transactions on Learning Technologies* 6.4, S. 364–377. ISSN: 1939-1382. DOI: 10.1109/tlt.2013.27.

„From Expert Systems to Intelligent Tutoring Systems" (1992). In: *Advanced Models of Cognition for Medical Training and Practice*. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 149–161. ISBN: 978-3-642-08144-6 978-3-662-02833-9. DOI: 10.1007/978-3-662-02833-9_8.

Fuller, Ursula et al. (Dez. 2007). „Developing a Computer Science-Specific Learning Taxonomy". In: *ACM SIGCSE Bulletin* 39.4, S. 152–170. ISSN: 0097-8418. DOI: 10.1145/1345375.1345438.

Futschek, Gerald (2006). „Algorithmic Thinking: The Key for Understanding Computer Science". In: *Informatics Education – The Bridge between Using and Understanding Computers*. Hrsg. von David Hutchison et al. Bd. 4226. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 159–168. ISBN: 978-3-540-48218-5 978-3-540-48227-7. DOI: 10.1007/11915355_15.

Gamboa, Hugo und Ana Fred (2001). „Designing Intelligent Tutoring Systems: A Bayesian Approach". In: *Proceedings of the 3rd International Conference on Enterprise Information Systems (ICEIS 2001)*. 3rd International Conference on Enterprise Information Systems (7.–10. Juli 2001). Hrsg. von Bernadette Sharp et al. Bd. 1. Setúbal, Portugal: ICEIS Press, S. 452–458. ISBN: 972-98050-2-4 978-972-98050-2-8.

Gardner, Howard (1987). *The Mind's New Science: A History of the Cognitive Revolution*. s.l.: BasicBooks. ISBN: 978-0-465-04635-5.

Gettier, E. L. (Juni 1963). „Is Justified True Belief Knowledge?" In: *Analysis* 23.6, S. 121–123. ISSN: 0003-2638, 1467-8284. DOI: 10.1093/analys/23.6.121.

Gläser-Zikuda, Michaela, Gerda Hagenauer und Melanie Stephan (Jan. 2020). „The Potential of Qualitative Content Analysis for Empirical Educational Research". In: *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research* Vol 21, No 1 (2020): Qualitative Content Analysis II. DOI: 10.17169/FQS-21.1.3443.

Göhner, Maximilian und Moritz Krell (Dez. 2020). „Qualitative Inhaltsanalyse in naturwissenschaftsdidaktischer Forschung unter Berücksichtigung von Gütekriterien: Ein Review". In: *Zeitschrift für Didaktik der Naturwissenschaften* 26.1, S. 207–225. ISSN: 0949-1147, 2197-988X. DOI: 10.1007/s40573-020-00111-0.

Haberbosch, Maximilian et al. (Juli 2025). „Entwicklung und Zusammenhang von Reflexionskompetenz und Selbstwirksamkeitserwartung im Lehr-Lern-Labor-Seminar". In: *Zeitschrift für Erziehungswissenschaft*. ISSN: 1434-663X, 1862-5215. DOI: 10.1007/s11618-025-01325-z.

Hafner, Rebecca, David Elmes und Daniel Read (Dez. 2019). „Exploring the Role of Messenger Effects and Feedback Frames in Promoting Uptake of Energy-Efficient Technologies". In: *Current Psychology* 38.6, S. 1601–1612. ISSN: 1046-1310, 1936-4733. DOI: 10.1007/s12144-017-9717-2.

Hahn, Larissa und Pascal Klein (Apr. 2025). „The Impact of Multiple Representations on Students' Understanding of Vector Field Concepts: Implementation of Simulations and Sketching Activities into Lecture-Based Recitations in Undergraduate Physics". In: *Frontiers in Psychology* 16, S. 1544764. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2025.1544764.

Hasan, Rakibul et al. (2025). „The Dark Side of AI Anthropomorphism: A Case of Misplaced Trustworthiness in Service Provisions". In: *Hawaii International Conference on System Sciences*. DOI: 10.24251/HICSS.2025.682. HDL: 10125/109530.

Hattie, John (2009). *Visible Learning: A Synthesis of over 800 Meta-Analyses Relating to Achievement*. London ; New York: Routledge. ISBN: 978-0-415-47618-8 978-0-415-47617-1.

Hattie, John und Helen Timperley (März 2007). „The Power of Feedback". In: *Review of Educational Research* 77.1, S. 81–112. ISSN: 0034-6543, 1935-1046. DOI: 10.3102/003465430298487.

Hellas, Arto et al. (Aug. 2023). „Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests". In: *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1*. Chicago IL USA: ACM, S. 93–105. ISBN: 978-1-4503-9976-0. DOI: 10.1145/3568813.3600139.

Helmke, Andreas (2022). *Unterrichtsqualität und Professionalisierung: Diagnostik von Lehr-Lern-Prozessen und evidenzbasierte Unterrichtsentwicklung*. 1. Auflage. Schule weiterentwickeln - Unterricht verbessern Orientierungsband. Hannover: Klett Kallmeyer. ISBN: 978-3-7800-1009-4 978-3-7727-1685-0 978-3-7727-1686-7 978-3-7727-1687-4.

Hevner, Alan (Jan. 2007). „A Three Cycle View of Design Science Research". In: *Scandinavian Journal of Information Systems* 19.

Hevner, Alan und Samir Chatterjee (2010). „Design Science Research in Information Systems". In: *Design Research in Information Systems*. Bd. 22. Boston, MA: Springer US, S. 9–22. ISBN: 978-1-4419-5652-1 978-1-4419-5653-8. DOI: 10.1007/978-1-4419-5653-8_2.

Hevner, Alan et al. (März 2004). „Design Science in Information Systems Research". In: *Management Information Systems Quarterly* 28, S. 75–.

Hill, Clara E., Barbara J. Thompson und Elizabeth Nutt Williams (Okt. 1997). „A Guide to Conducting Consensual Qualitative Research". In: *The Counseling Psychologist* 25.4, S. 517–572. ISSN: 0011-0000, 1552-3861. DOI: 10.1177/0011000097254001.

Hill, Clara E. et al. (Apr. 2005). „Consensual Qualitative Research: An Update." In: *Journal of Counseling Psychology* 52.2, S. 196–205. ISSN: 1939-2168, 0022-0167. DOI: 10.1037/0022-0167.52.2.196.

Hilpert, Jonathan C. und Gwen C. Marchand (Juli 2018). „Complex Systems Research in Educational Psychology: Aligning Theory and Method". In: *Educational Psychologist* 53.3, S. 185–202. ISSN: 0046-1520, 1532-6985. DOI: 10.1080/00461520.2018.1469411.

Homt, Martina und Bea Bloh (März 2025). „Evidenzorientierung im Referendariat – Analyse der Lern- und Handlungspraxis auf Basis einer Evidenzsystematik". In: *Unterrichtswissenschaft*. ISSN: 0340-4099, 2520-873X. DOI: 10.1007/s42010-025-00222-y.

Howard, Jeremy und Sebastian Ruder (Mai 2018). *Universal Language Model Fine-tuning for Text Classification*. DOI: 10.48550/arXiv.1801.06146. arXiv: 1801.06146 [cs].

Hsu, Hsiao-Ping (Mai 2025). „From Programming to Prompting: Developing Computational Thinking through Large Language Model-Based Generative Artificial Intelligence". In: *TechTrends* 69.3, S. 485–506. ISSN: 8756-3894, 1559-7075. DOI: 10.1007/s11528-025-01052-6.

Hubwieser, Peter et al. (Nov. 2013). „Pedagogical Content Knowledge for Computer Science in German Teacher Education Curricula". In: *Proceedings of the 8th Workshop in Primary and Secondary Computing Education*. Aarhus Denmark: ACM, S. 95–103. ISBN: 978-1-4503-2455-7. DOI: 10.1145/2532748.2532753.

Ihantola, Petri et al. (Okt. 2010). „Review of Recent Systems for Automatic Assessment of Programming Assignments". In: *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. Koli Finland: ACM, S. 86–93. ISBN: 978-1-4503-0520-4. DOI: 10.1145/1930464.1930480.

Ihantola, Petri et al. (Juli 2015). „Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies". In: *Proceedings of the 2015 ITiCSE on Working Group Reports*. Vilnius Lithuania: ACM, S. 41–63. ISBN: 978-1-4503-4146-2. DOI: 10.1145/2858796.2858798.

Iivari, Juhani (Jan. 2007). „A Paradigmatic Analysis of Information Systems as a Design Science". In: *Scandinavian Journal of Information Systems* 19, S. 39–.

Ilgen, Daniel R., Cynthia D. Fisher und M. Susan Taylor (Aug. 1979). „Consequences of Individual Feedback on Behavior in Organizations." In: *Journal of Applied Psychology* 64.4, S. 349–371. ISSN: 1939-1854, 0021-9010. DOI: 10.1037/0021-9010.64.4.349.

Ishizue, Ryosuke et al. (März 2024). „Improved Program Repair Methods Using Refactoring with GPT Models". In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. Portland OR USA: ACM, S. 569–575. ISBN: 979-8-4007-0423-9. DOI: 10.1145/3626252.3630875.

Islam, Md. Mahmudul et al. (Juni 2025). „The Integration of Explainable AI in Educational Data Mining for Student Academic Performance Prediction and Support System". In: *Telematics and Informatics Reports* 18, S. 100203. ISSN: 27725030. DOI: 10.1016/j.teler.2025.100203.

Jacobs, Bernhard (2002). „Aufgaben stellen und Feedback geben". In: DOI: 10.23668/PSYCHARCHIVES.8555.

Jacobs, Sven und Steffen Jaschke (Juli 2024). „Leveraging Lecture Content for Improved Feedback: Explorations with GPT-4 and Retrieval Augmented Generation". In: *2024 36th International Conference on Software Engineering Education and Training (CSEE&T)*. Würzburg, Germany: IEEE, S. 1–5. ISBN: 979-8-3503-7897-9. DOI: 10.1109/CSEET62301.2024.10663001.

Jeuring, Johan et al. (Dez. 2022). „Towards Giving Timely Formative Feedback and Hints to Novice Programmers". In: *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education*. Dublin Ireland: ACM, S. 95–115. ISBN: 979-8-4007-0010-1. DOI: 10.1145/3571785.3574124.

Ji, Shaoxiong et al. (Feb. 2022). „A Survey on Knowledge Graphs: Representation, Acquisition, and Applications". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.2, S. 494–514. ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2021.3070843.

Johnson, Colin G. und Ursula Fuller (Feb. 2006). „Is Bloom's Taxonomy Appropriate for Computer Science?" In: *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006*. Uppsala Sweden: ACM, S. 120–123. ISBN: 978-1-4503-7838-3. DOI: 10.1145/1315803.1315825.

Joshi, Ishika et al. (März 2024). „ChatGPT in the Classroom: An Analysis of Its Strengths and Weaknesses for Solving Undergraduate Computer Science Questions". In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. Portland OR USA: ACM, S. 625–631. ISBN: 979-8-4007-0423-9. DOI: 10.1145/3626252.3630803.

Kazemitabaar, Majeed et al. (Apr. 2023). „Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming". In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. Hamburg Germany: ACM, S. 1–23. ISBN: 978-1-4503-9421-5. DOI: 10.1145/3544548.3580919.

Kerres, Michael und Claudia de Witt (2004). „Pragmatismus Als Theoretische Grundlage Zur Konzeption von eLearning". In: *Handlungsorientiertes Lernen Und eLearning: Grundlagen Und Beispiele*. Hrsg. von D. Treichel und H. O. Meyer. München: Oldenbourg Verlag, S. 1–17.

Keuning, Hieke (2020). „Automated Feedback for Learning Code Refactoring". Diss. Heerlen: Open Universiteit.

Keuning, Hieke, Johan Jeuring und Bastiaan Heeren (Juli 2016). „Towards a Systematic Review of Automated Feedback Generation for Programming Exercises". In: *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. Arequipa Peru: ACM, S. 41–46. ISBN: 978-1-4503-4231-5. DOI: 10.1145/2899415.2899422.

Keuning, Hieke, Johan Jeuring und Bastiaan Heeren (März 2019). „A Systematic Literature Review of Automated Feedback Generation for Programming Exercises". In: *ACM Transactions on Computing Education* 19.1, S. 1–43. ISSN: 1946-6226. DOI: 10.1145/3231711.

Keuning, Hieke et al. (Nov. 2024). „Students' Perceptions and Use of Generative AI Tools for Programming Across Different Computing Courses". In: *Proceedings of the 24th Koli Calling International Conference on Computing Education Research*. Koli Finland: ACM, S. 1–12. ISBN: 979-8-4007-1038-4. DOI: 10.1145/3699538.3699546.

Khosravi, Hamed et al. (2024). „Chatbots and ChatGPT: A Bibliometric Analysis and Systematic Review of Publications in Web of Science and Scopus Databases". In: *International Journal of Data Mining, Modelling and Management* 16.2, S. 113–147. ISSN: 1759-1163, 1759-1171. DOI: 10.1504/IJDMMM.2024.138824.

Kierner, Slawomir, Jacek Kucharski und Zofia Kierner (Aug. 2023). „Taxonomy of Hybrid Architectures Involving Rule-Based Reasoning and Machine Learning in Clinical Decision Systems: A Scoping Review". In: *Journal of Biomedical Informatics* 144, S. 104428. ISSN: 15320464. DOI: 10.1016/j.jbi.2023.104428.

Kiesler, Natalie (Nov. 2020a). „On Programming Competence and Its Classification". In: *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*. Koli Finland: ACM, S. 1–10. ISBN: 978-1-4503-8921-1. DOI: 10.1145/3428029.3428030.

Kiesler, Natalie (Juni 2020b). „Towards a Competence Model for the Novice Programmer Using Bloom's Revised Taxonomy - An Empirical Approach". In: *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. Trondheim Norway: ACM, S. 459–465. ISBN: 978-1-4503-6874-2. DOI: 10.1145/3341525.3387419.

Kiesler, Natalie (2021). „Kompetenzförderung in der Programmierausbildung durch Modellierung von Kompetenzen und informativem Feedback". Diss. Frankfurt am Main: Johann Wolfgang Goethe-Universität.

Kiesler, Natalie (Juni 2022). *Daten- Und Methodenbericht Rekursive Problemlösung in Der Online Lernumgebung CodingBat Durch Informatik-Studierende*.

Kiesler, Natalie (2024). *Modeling Programming Competency: A Qualitative Analysis*. Cham: Springer International Publishing. ISBN: 978-3-031-47147-6 978-3-031-47148-3. DOI: 10.1007/978-3-031-47148-3.

Kiesler, Natalie, Goethe-Universität Frankfurt Am Main und Hochschule Fulda (2022). *Recursive Problem Solving in the Online Learning Environment CodingBat by Computer Science studentsRekursive Problemlösung in Der Online Lernumgebung CodingBat Durch Informatik-Studierende*. DOI: 10.21249/DZHW:STUDENTSTEPS:1.0.0.

Kiesler, Natalie, Dominic Lohr und Hieke Keuning (Okt. 2023). „Exploring the Potential of Large Language Models to Generate Formative Programming Feedback". In: *2023 IEEE Frontiers in Education Conference (FIE)*. College Station, TX, USA: IEEE, S. 1–5. ISBN: 979-8-3503-3642-9. DOI: 10.1109/FIE58773.2023.10343457.

Kiesler, Natalie und Daniel Schiffner (2023a). *Large Language Models in Introductory Programming Education: ChatGPT's Performance and Implications for Assessments*. DOI: 10.48550/ARXIV.2308.08572.

Kiesler, Natalie und Daniel Schiffner (Juni 2023b). „Why We Need Open Data in Computer Science Education Research". In: *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. Turku Finland: ACM, S. 348–353. ISBN: 979-8-4007-0138-2. DOI: 10.1145/3587102.3588860.

Kimmel, Bailey et al. (Mai 2024). „Enhancing Programming Error Messages in Real Time with Generative AI". In: *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. Honolulu HI USA: ACM, S. 1–7. ISBN: 979-8-4007-0331-7. DOI: 10.1145/3613905.3647967.

Klauer, Karl Josef (1987). *Kriteriumsorientierte Tests: Lehrbuch Der Theorie Und Praxis Lehrzielorientierten Messens*. Göttingen: C.J. Hogrefe. ISBN: 978-3-8017-0256-4.

Kleinknecht, Marc et al., Hrsg. (2013). *Lern- und Leistungsaufgaben im Unterricht: fächerübergreifende Kriterien zur Auswahl und Analyse*. Bad Heilbrunn: Verlag Julius Klinkhardt. ISBN: 978-3-7815-1930-5.

Kluger, Avraham N. und Angelo DeNisi (März 1996). „The Effects of Feedback Interventions on Performance: A Historical Review, a Meta-Analysis, and a Preliminary Feedback Intervention Theory." In: *Psychological Bulletin* 119.2, S. 254–284. ISSN: 1939-1455, 0033-2909. DOI: 10.1037/0033-2909.119.2.254.

Koedinger, Kenneth et al. (Feb. 1997). „Intelligent Tutoring Goes to School in the Big City". In: *International Journal of Artificial Intelligence in Education* 8, S. 30–43.

Kohlhase, Michael (Dez. 2008). „Using LaTeX as a Semantic Markup Format". In: *Mathematics in Computer Science* 2.2, S. 279–304. ISSN: 1661-8270, 1661-8289. DOI: 10.1007/s11786-008-0055-5.

Körndle, Hermann, Susanne Narciss und Antje Proske (2004). „Konstruktion interaktiver Lernaufgaben für die universitäre Lehre". In: *Campus 2004. Kommen die digitalen Medien an den Hochschulen in die Jahre?* Bd. Medien in der Wissenschaft. Waxmann : Münster u. a., S. 57–67. ISBN: 978-3-8309-1417-4. DOI: 10.25656/01:11265.

Koutcheme, Charles et al. (Juli 2024). „Open Source Language Models Can Provide Feedback: Evaluating LLMs' Ability to Help Students Using GPT-4-As-A-Judge". In: *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. Milan Italy: ACM, S. 52–58. ISBN: 979-8-4007-0600-4. DOI: 10.1145/3649217.3653612.

Krause, Ulrike-Marie (2007). *Feedback und kooperatives Lernen*. Pädagogische Psychologie und Entwicklungspsychologie 60. Münster: Waxmann. ISBN: 978-3-8309-1806-6.

Kuckartz, Udo (2018). *Qualitative Inhaltsanalyse. Methoden, Praxis, Computerunterstützung*. 4., überarbeitete Aufl. Grundlagentexte Methoden. Weinheim: Beltz. ISBN: 978-3-7799-3682-4 978-3-7799-4683-0.

Kulhavy, Raymond W. und William A. Stock (Dez. 1989). „Feedback in Written Instruction: The Place of Response Certitude". In: *Educational Psychology Review* 1.4, S. 279–308. ISSN: 1040-726X, 1573-336X. DOI: 10.1007/BF01320096.

Kulik, James A. und J. D. Fletcher (März 2016). „Effectiveness of Intelligent Tutoring Systems: A Meta-Analytic Review". In: *Review of Educational Research* 86.1, S. 42–78. ISSN: 0034-6543, 1935-1046. DOI: 10.3102/0034654315581420.

Kulik, James A. und Chen-Lin C. Kulik (1988). „Timing of Feedback and Verbal Learning". In: *Review of Educational Research* 58.1, S. 79. ISSN: 00346543. DOI: 10.2307/1170349.

Lahtinen, Essi, Kirsti Ala-Mutka und Hannu-Matti Järvinen (Sep. 2005). „A Study of the Difficulties of Novice Programmers". In: *ACM SIGCSE Bulletin* 37.3, S. 14–18. ISSN: 0097-8418. DOI: 10.1145/1151954.1067453.

Lämmel, Uwe (2020). *Künstliche Intelligenz: Wissensverarbeitung - neuronale Netze*. 5., überarbeitete Auflage. Hanser eLibrary. München: Hanser. ISBN: 978-3-446-45914-4 978-3-446-46363-9. DOI: 10.3139/9783446463639.

Landis, J. Richard und Gary G. Koch (März 1977). „The Measurement of Observer Agreement for Categorical Data". In: *Biometrics* 33.1, S. 159. ISSN: 0006341X. DOI: 10.2307/2529310. JSTOR: 2529310.

Le, Nguyen-Thinh (Mai 2016). „A Classification of Adaptive Feedback in Educational Systems for Programming". In: *Systems* 4.2, S. 22. ISSN: 2079-8954. DOI: 10.3390/systems4020022.

Lehtinen, Teemu (2022). *FITech Dataset*.

Leinonen, Juho et al. (Juni 2023a). „Comparing Code Explanations Created by Students and Large Language Models". In: *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. Turku Finland: ACM, S. 124–130. ISBN: 979-8-4007-0138-2. DOI: 10.1145/3587102.3588785.

Leinonen, Juho et al. (März 2023b). „Using Large Language Models to Enhance Programming Error Messages". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science*

Education V. 1. Toronto ON Canada: ACM, S. 563–569. ISBN: 978-1-4503-9431-4. DOI: 10.1145/3545945.3569770.

Lewis, Patrick et al. (2020). „Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. Nips '20. Red Hook, NY, USA: Curran Associates Inc. ISBN: 978-1-7138-2954-6.

Lindland, O.I., G. Sindre und A. Solvberg (März 1994). „Understanding Quality in Conceptual Modeling". In: *IEEE Software* 11.2, S. 42–49. ISSN: 0740-7459. DOI: 10.1109/52.268955.

Lohr, Dominic und Marc Berges (Sep. 2021). „Towards Criteria for Valuable Automatic Feedback in Large Programming Classes". In: *Hochschuldidaktik Informatik HDI 2021*. Dortmund: Jörg Desel, Simone Opel, S. 181–186. ISBN: 978-3-00-070267-9.

Lohr, Dominic und Marc Berges (2025). „Der Weg ist das Ziel – Ein skalierbarer Wechsel von summativem zu formativem Assessment in der Programmierausbildung mit Antwortklassen". In: *Proceedings Seventh Workshop "Automatische Bewertung von Programmieraufgaben"(ABP 2025)*. Gesellschaft für Informatik e.V. DOI: 10.18420/ABP2025_05.

Lohr, Dominic, Hieke Keuning und Natalie Kiesler (Feb. 2025). „You're (Not) My Type – Can LLMs Generate Feedback of Specific Types for Introductory Programming Tasks?" In: *Journal of Computer Assisted Learning* 41.1, e13107. ISSN: 0266-4909, 1365-2729. DOI: 10.1111/jcal.13107.

Lohr, Dominic, Lars Wechsler und Marc Berges (Juni 2025). „Introducing a Self-Study Course for Learning Textual Programming at Highschool with APFEL and ALeA". In: *Workshop KI Und Bildung 2024*. DOI: 10.20378/irb-108298.

Lohr, Dominic et al. (Aug. 2023). „The Y-Model - Formalization of Computer Science Tasks in the Context of Adaptive Learning Systems". In: *2023 IEEE 2nd German Education Conference (GECon)*. Berlin, Germany: IEEE, S. 1–6. ISBN: 979-8-3503-4813-2. DOI: 10.1109/GECon58119.2023.10295148.

Lohr, Dominic et al. (Juli 2024a). „"Let Them Try to Figure It Out First" - Reasons Why Experts (Do Not) Provide Feedback to Novice Programmers". In: *Proceedings of the 2024 Innovation and Technology in Computer Science Education (ITiCSE 2024)*. Bd. 1. Milan, Italy: ACM, S. 7. DOI: 10.1145/3649217.3653530.

Lohr, Dominic et al. (2024b). „Adaptive Learning Systems in Programming Education: A Prototype for Enhanced Formative Feedback". In: *Proceedings of DELFI 2024*. Fulda: Gesellschaft für Informatik e.V., S. 549–554. DOI: 10.18420/DELFI2024_57.

Lohr, Dominic et al. (Okt. 2025a). „"Ignore These Errors for Now- How Experts Provide Feedback on Steps Novices Take Towards Solving Programming Problems". In: *Proceedings of the ACM Global on Computing Education Conference 2025 Vol 1*. Gaborone Botswana: ACM, S. 149–155. ISBN: 979-8-4007-1929-5. DOI: 10.1145/3736181.3747150.

Lohr, Dominic et al. (Feb. 2025b). „Leveraging Large Language Models to Generate Course-Specific Semantically Annotated Learning Objects". In: *Journal of Computer Assisted Learning* 41.1, e13101. ISSN: 0266-4909, 1365-2729. DOI: 10.1111/jcal.13101.

Lohr, Dominic et al. (2025c). „The Potential of Answer Classes in Large-scale Written Computer-Science Exams – Vol.2". In: *Commentarii informaticae didacticae*. Aachen: Jörg Desel, Simone Opel. DOI: 10.25932/publishup-68780.

Lunze, Jan (2010). „Regelungstechnik. 1: Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen: mit 413 Abbildungen, 75 Beispielen, 165 Übungsaufgaben sowie einer Einführung in das Programmsystem MATLAB". In: 8., neu bearb. Aufl. Springer-Lehrbuch. Berlin: Springer. ISBN: 978-3-642-13808-9.

Lyulina, Elena et al. (März 2021). „TaskTracker-tool: A Toolkit for Tracking of Code Snapshots and Activity Data During Solution of Programming Tasks". In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. Virtual Event USA: ACM, S. 495–501. ISBN: 978-1-4503-8062-1. DOI: 10.1145/3408877.3432534.

Maclean, Johanna Catherine, John Buckell und Joachim Marti (März 2019). *Information Source and Cigarettes: Experimental Evidence on the Messenger Effect*. Techn. Ber. w25632. Cambridge, MA: National Bureau of Economic Research, w25632. DOI: 10.3386/w25632.

MacNeil, Stephen et al. (März 2022a). „Automatically Generating CS Learning Materials with Large Language Models". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*. Toronto ON Canada: ACM, S. 1176–1176. ISBN: 978-1-4503-9433-8. DOI: 10.1145/3545947.3569630.

MacNeil, Stephen et al. (Aug. 2022b). „Generating Diverse Code Explanations Using the GPT-3 Large Language Model". In: *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 2*. Lugano und Virtual Event Switzerland: ACM, S. 37–39. ISBN: 978-1-4503-9195-5. DOI: 10.1145/3501709.3544280.

MacNeil, Stephen et al. (März 2023). „Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. Toronto ON Canada: ACM, S. 931–937. ISBN: 978-1-4503-9431-4. DOI: 10.1145/3545945.3569785.

Maeda, Takuya (Jan. 2025). „Misplaced Capabilities: Evaluating the Risks of Anthropomorphism in Human-AI Interactions". In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society* 7.2, S. 35–36. ISSN: 3065-8365. DOI: 10.1609/aies.v7i2.31903.

Maier, Uwe und Christian Klotz (2022). „Personalized Feedback in Digital Learning Environments: Classification Framework and Literature Review". In: *Computers and Education: Artificial Intelligence* 3, S. 100080. ISSN: 2666920X. DOI: 10.1016/j.caeai.2022.100080.

March, Salvatore T. und Gerald F. Smith (Dez. 1995). „Design and Natural Science Research on Information Technology". In: *Decision Support Systems* 15.4, S. 251–266. ISSN: 01679236. DOI: 10.1016/0167-9236(94)00041-2.

Marchegiani, Beatrice (Apr. 2025). „Anthropomorphism, False Beliefs, and Conversational AIs : How Chatbots Undermine Users' Autonomy". In: *Journal of Applied Philosophy*, japp.70008. ISSN: 0264-3758, 1468-5930. DOI: 10.1111/japp.70008.

Margulieux, Lauren E., Richard Catrambone und Mark Guzdial (Jan. 2016). „Employing Subgoals in Computer Programming Education". In: *Computer Science Education* 26.1, S. 44–67. ISSN: 0899-3408, 1744-5175. DOI: 10.1080/08993408.2016.1144429.

Mathan, Santosh A. und Kenneth R. Koedinger (Sep. 2005). „Fostering the Intelligent Novice: Learning From Errors With Metacognitive Tutoring". In: *Educational Psychologist* 40.4, S. 257–265. ISSN: 0046-1520, 1532-6985. DOI: 10.1207/s15326985ep4004_7.

Mayer, Richard E. (Apr. 2001). *Multimedia Learning*. 1. Aufl. Cambridge University Press. ISBN: 978-0-521-78239-5 978-0-521-78749-9 978-1-139-16460-3. DOI: 10.1017/CBO9781139164603.

Mayor, Adrienne (Dez. 2019). „Daedalus and the Living Statues". In: *Gods and Robots: Daedalus and the Living Statues*. Princeton University Press, S. 85–104. ISBN: 978-0-691-18544-6. DOI: 10.1515/9780691185446-008.

Mayring, Philipp (2014). *Qualitative Content Analysis: Theoretical Foundation, Basic Procedures and Software Solution*. Klagenfurt: gesis Leibniz-Institut für Sozialwissenschaft, S. 143.

Mayring, Philipp (2015a). „Qualitative Content Analysis: Theoretical Background and Procedures". In: *Approaches to Qualitative Research in Mathematics Education*. Hrsg. von Angelika Bikner-Ahsbahs, Christine Knipping und Norma Presmeg. Dordrecht: Springer Netherlands, S. 365–380. ISBN: 978-94-017-9180-9 978-94-017-9181-6. DOI: 10.1007/978-94-017-9181-6_13.

Mayring, Philipp (2015b). *Qualitative Inhaltsanalyse: Grundlagen und Techniken*. 12., überarbeitete Auflage. Weinheim Basel: Beltz. ISBN: 978-3-407-25730-7.

Mayring, Philipp (Jan. 2025). „Qualitative Inhaltsanalyse mit ChatGPT: Fallstricke, grobe Annäherungen und grobe Fehler. Ein Erfahrungsbericht". In: *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research* 26.1. DOI: 10.17169/FQS-26.1.4252.

McCall, Davin und Michael Kölling (Dez. 2019). „A New Look at Novice Programmer Errors". In: *ACM Transactions on Computing Education* 19.4, S. 1–30. ISSN: 1946-6226. DOI: 10.1145/3335814.

McCarthy, John et al. (Dez. 2006). „A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955". In: *AI Magazine* 27.4, S. 12. DOI: 10.1609/aimag.v27i4.1904.

McGill, Tanya J. und Simone E. Volet (März 1997). „A Conceptual Framework for Analyzing Students' Knowledge of Programming". In: *Journal of Research on Computing in Education* 29.3, S. 276–297. ISSN: 0888-6504. DOI: 10.1080/08886504.1997.10782199.

Menon, Tanya und Sally Blount (Jan. 2003). „The Messenger Bias: A Relational Model of Knowledge Valuation". In: *Research in Organizational Behavior* 25, S. 137–186. ISSN: 01913085. DOI: 10.1016/S0191-3085(03)25004-8.

Merrill, Douglas C. et al. (Juli 1992). „Effective Tutoring Techniques: A Comparison of Human Tutors and Intelligent Tutoring Systems". In: *Journal of the Learning Sciences* 2.3, S. 277–305. ISSN: 1050-8406, 1532-7809. DOI: 10.1207/s15327809jls0203_2.

Merrill, M. David (Sep. 2002). „First Principles of Instruction". In: *Educational Technology Research and Development* 50.3, S. 43–59. ISSN: 1042-1629, 1556-6501. DOI: 10.1007/BF02505024.

Messer, Marcus et al. (März 2024). „Automated Grading and Feedback Tools for Programming Education: A Systematic Review". In: *ACM Transactions on Computing Education* 24.1, S. 1–43. ISSN: 1946-6226. DOI: 10.1145/3636515.

Mitrovic, Antonija (2010). „Modeling Domains and Students with Constraint-Based Modeling". In: *Advances in Intelligent Tutoring Systems*. Hrsg. von Janusz Kacprzyk et al. Bd. 308. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 63–80. ISBN: 978-3-642-14362-5 978-3-642-14363-2. DOI: 10.1007/978-3-642-14363-2_4.

Mitrovic, Antonija (Apr. 2012). „Fifteen Years of Constraint-Based Tutors: What We Have Achieved and Where We Are Going". In: *User Modeling and User-Adapted Interaction* 22.1-2, S. 39–72. ISSN: 0924-1868, 1573-1391. DOI: 10.1007/s11257-011-9105-9.

Müller, Dennis (Mai 2026). „The Flexiformalist Manifesto Made Manifest". In: *Proceedings of the 2026 International Conference on Breakthroughs in Artificial Intelligence (BAI'26)*. Lecture Notes in Networks and Systems. Tunis, Tunisia: Springer. (accepted for publication).

Narciss, Susanne (2006). *Informatives tutorielles Feedback: Entwicklungs- und Evaluationsprinzipien auf der Basis instruktionspsychologischer Erkenntnisse*. Pädagogische Psychologie und Entwicklungspsychologie 56. Münster New York München Berlin: Waxmann. ISBN: 978-3-8309-1641-3.

Narciss, Susanne (Dez. 2007). „Feedback Strategies for Interactive Learning Tasks". In: *Handbook of Research on Educational Communications and Technology*. 3. Aufl. Routledge, S. 19. ISBN: 978-0-203-88086-9.

Narciss, Susanne (2012). „Feedback Strategies". In: *Encyclopedia of the Sciences of Learning*. Hrsg. von Norbert M. Seel. Boston, MA: Springer US, S. 1289–1293. ISBN: 978-1-4419-1427-9 978-1-4419-1428-6. DOI: 10.1007/978-1-4419-1428-6_283.

Narciss, Susanne (2017). „Conditions and Effects of Feedback Viewed Through the Lens of the Interactive Tutoring Feedback Model". In: *Scaling up Assessment for Learning in Higher Education*. Hrsg. von David Carless et al. Bd. 5. Singapore: Springer Singapore, S. 173–189. ISBN: 978-981-10-3043-7 978-981-10-3045-1. DOI: 10.1007/978-981-10-3045-1_12.

Narciss, Susanne (2018). „Feedbackstrategien für interaktive Lernaufgaben". In: *Praxishandbuch Professionelle Mediation*. Hrsg. von Stefan Kracht, Andre Niedostadek und Patrick Sensburg. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 1–24. ISBN: 978-3-662-49657-2. DOI: 10.1007/978-3-662-54373-3_35-1.

Narciss, Susanne (2020). „Feedbackstrategien für interaktive Lernaufgaben". In: *Handbuch Bildungstechnologie*. Hrsg. von Helmut Niegemann und Armin Weinberger. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 369–392. ISBN: 978-3-662-54367-2 978-3-662-54368-9. DOI: 10.1007/978-3-662-54368-9_35.

Narciss, Susanne und Katja Huth (Aug. 2006). „Fostering Achievement and Motivation with Bug-Related Tutoring Feedback in a Computer-Based Training for Written Subtraction". In: *Learning and Instruction* 16.4, S. 310–322. ISSN: 09594752. DOI: 10.1016/j.learninstruc.2006.07.003.

Neisser, Ulric (1967). *Cognitive Psychology*. Century Psychology Series. Englewood Cliffs, N.J: Prentice Hall. ISBN: 978-0-13-139667-8.

Neuwinger, Max und Dirk Riehle (2025). *A Systematic Review of Common Beginner Programming Mistakes in Data Engineering*. DOI: 10.48550/ARXIV.2504.16644.

Nguyen, Ha und Vicki Allan (März 2024). „Using GPT-4 to Provide Tiered, Formative Code Feedback". In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. Portland OR USA: ACM, S. 958–964. ISBN: 979-8-4007-0423-9. DOI: 10.1145/3626252.3630960.

Nicol, David J. und Debra Macfarlane-Dick (Apr. 2006). „Formative Assessment and Self-regulated Learning: A Model and Seven Principles of Good Feedback Practice". In: *Studies in Higher Education* 31.2, S. 199–218. ISSN: 0307-5079, 1470-174X. DOI: 10.1080/03075070600572090.

Nkambou, Roger, Jacqueline Bourdeau und Riichiro Mizoguchi, Hrsg. (2010). *Advances in Intelligent Tutoring Systems*. Studies in Computational Intelligence. Berlin, Heidelberg: Springer

Berlin Heidelberg. ISBN: 978-3-642-14362-5 978-3-642-14363-2. DOI: 10.1007/978-3-642-14363-2.

Nwana, HyacinthS. (1990). „Intelligent Tutoring Systems: An Overview". In: *Artificial Intelligence Review* 4.4. ISSN: 0269-2821, 1573-7462. DOI: 10.1007/BF00168958.

Ohlsson, Stellan (1994). „Constraint-Based Student Modeling". In: *Student Modelling: The Key to Individualized Knowledge-Based Instruction*. Hrsg. von Jim E. Greer und Gordon I. McCalla. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 167–189. ISBN: 978-3-642-08186-6 978-3-662-03037-0. DOI: 10.1007/978-3-662-03037-0_7.

Okpo, Juliet (Juli 2016). „Adaptive Exercise Selection for an Intelligent Tutoring System". In: *Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization*. Halifax Nova Scotia Canada: ACM, S. 313–316. ISBN: 978-1-4503-4368-8. DOI: 10.1145/2930238.2930369.

OpenAI (2023). „GPT-4 Technical Report". In: S. 100. DOI: 10.48550/ARXIV.2303.08774.

OpenAI (2024). *OpenAI API Documentation: Prompt Engineering*.

OpenAI et al. (März 2024). *GPT-4 Technical Report*. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 [cs].

Padayachee, Indira (Jan. 2002). „Intelligent Tutoring Systems: Architecture and Characteristics". In: *Proceedings of the 32nd Annual SACLA Conference, 2002*.

Paiva, José Carlos, José Paulo Leal und Álvaro Figueira (Sep. 2022). „Automated Assessment in Computer Science Education: A State-of-the-Art Review". In: *ACM Transactions on Computing Education* 22.3, S. 1–40. ISSN: 1946-6226, 1946-6226. DOI: 10.1145/3513140.

Panadero, Ernesto und Anastasiya A. Lipnevich (Feb. 2022). „A Review of Feedback Models and Typologies: Towards an Integrative Model of Feedback Elements". In: *Educational Research Review* 35, S. 100416. ISSN: 1747938X. DOI: 10.1016/j.edurev.2021.100416.

Pankiewicz, Maciej und Ryan S. Baker (2023). „Large Language Models (GPT) for Automating Feedback on Programming Assignments". In: DOI: 10.48550/ARXIV.2307.00150.

Paramythis, Alexandros und Susanne Loidl-Reisinger (2004). „Adaptive Learning Environments and e-Learning Standards". In: *Electronic Journal of E-Learning* 2.1, S. 181–194.

Pastor, Oscar (2008). „Conceptual Modeling Meets the Human Genome". In: *Conceptual Modeling – ER 2008*. Hrsg. von Qing Li et al. Bd. 5231. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, S. 1–11. DOI: 10.1007/978-3-540-87877-3_1.

Paulson Gjerde, Kathy, Margaret Y. Padgett und Deborah Skinner (2017). „The Impact of Process vs. Outcome Feedback on Student Performance and Perceptions". In: *Journal of Learning in Higher Education* 13.1, S. 73–82.

Perrigo, Billy (Jan. 2023). *Exclusive: The $2 Per Hour Workers Who Made ChatGPT Safer*. https://time.com/6247678/openai-chatgpt-kenya-workers/.

Petrina, Stephen (Apr. 2004). „Sidney Pressey and the Automation of Education, 1924-1934". In: *Technology and Culture* 45.2, S. 305–330. ISSN: 1097-3729. DOI: 10.1353/tech.2004.0085.

Phung, Tung et al. (März 2024). „Automating Human Tutor-Style Programming Feedback: Leveraging GPT-4 Tutor Model for Hint Generation and GPT-3.5 Student Model for Hint

Validation". In: *Proceedings of the 14th Learning Analytics and Knowledge Conference*. Kyoto Japan: ACM, S. 12–23. ISBN: 979-8-4007-1618-8. DOI: 10.1145/3636555.3636846.

Piaget, Jean (Sep. 2003). *The Psychology of Intelligence*. 0. Aufl. Routledge. ISBN: 978-1-134-52469-3. DOI: 10.4324/9780203164730.

Poldrack, R (Feb. 2006). „Can Cognitive Processes Be Inferred from Neuroimaging Data?" In: *Trends in Cognitive Sciences* 10.2, S. 59–63. ISSN: 13646613. DOI: 10.1016/j.tics.2005.12.004.

Prather, James et al. (Dez. 2023a). „The Robots Are Here: Navigating the Generative AI Revolution in Computing Education". In: *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education*. Turku Finland: ACM, S. 108–159. ISBN: 979-8-4007-0405-5. DOI: 10.1145/3623762.3633499.

Prather, James et al. (Juni 2023b). „Transformed by Transformers: Navigating the AI Coding Revolution for Computing Education: An ITiCSE Working Group Conducted by Humans". In: *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 2*. Turku Finland: ACM, S. 561–562. ISBN: 979-8-4007-0139-9. DOI: 10.1145/3587103.3594206.

Prather, James et al. (Juli 2024). „How Instructors Incorporate Generative AI into Teaching Computing". In: *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 2*. Milan Italy: ACM, S. 771–772. ISBN: 979-8-4007-0603-5. DOI: 10.1145/3649405.3659534.

Prather, James et al. (Jan. 2025). „Beyond the Hype: A Comprehensive Review of Current Trends in Generative AI Research, Teaching Practices, and Tools". In: *2024 Working Group Reports on Innovation and Technology in Computer Science Education*. Milan Italy: ACM, S. 300–338. ISBN: 979-8-4007-1208-1. DOI: 10.1145/3689187.3709614.

Psotka, Joseph, L. Daniel Massey und Sharon A. Mutter, Hrsg. (1988). *Intelligent Tutoring Systems: Lessons Learned*. Hillsdale, N.J: L. Erlbaum Associates. ISBN: 978-0-8058-0023-4 978-0-8058-0192-7.

Qian, Yizhou und James Lehman (März 2018). „Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review". In: *ACM Transactions on Computing Education* 18.1, S. 1–24. ISSN: 1946-6226. DOI: 10.1145/3077618.

Radford, Alec et al. (2018). *Improving Language Understanding by Generative Pre-Training*. Techn. Ber. OpenAI.

Ramaprasad, Arkalgud (Jan. 1983). „On the Definition of Feedback". In: *Behavioral Science* 28.1, S. 4–13. ISSN: 00057940, 10991743. DOI: 10.1002/bs.3830280103.

Rebedea, Traian et al. (2023). „NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications with Programmable Rails". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Singapore: Association for Computational Linguistics, S. 431–445. DOI: 10.18653/v1/2023.emnlp-demo.40.

Reddy, Y. Malini und Heidi Andrade (Juli 2010). „A Review of Rubric Use in Higher Education". In: *Assessment & Evaluation in Higher Education* 35.4, S. 435–448. ISSN: 0260-2938, 1469-297X. DOI: 10.1080/02602930902862859.

Rich, Elaine (1988). *Artificial Intelligence*. Internat. ed., 8. print. McGraw-Hill Series in Artificial Intelligence. Auckland: McGraw-Hill. ISBN: 978-0-07-052261-9.

Robins, Anthony, Janet Rountree und Nathan Rountree (Juni 2003). „Learning and Teaching Programming: A Review and Discussion". In: *Computer Science Education* 13.2, S. 137–172. ISSN: 0899-3408, 1744-5175. DOI: 10.1076/csed.13.2.137.14200.

Roest, Lianne, Hieke Keuning und Johan Jeuring (Jan. 2024). „Next-Step Hint Generation for Introductory Programming Using Large Language Models". In: *Proceedings of the 26th Australasian Computing Education Conference*. Sydney NSW Australia: ACM, S. 144–153. ISBN: 979-8-4007-1619-5. DOI: 10.1145/3636243.3636259.

Roethlisberger, Fritz J. et al., Hrsg. (2003). *Management and the Worker*. The Early Sociology of Management and Organizations v. 5. London: Routledge. ISBN: 978-0-415-27987-1 978-0-415-27982-6.

Roll, Ido et al. (Apr. 2011). „Improving Students' Help-Seeking Skills Using Metacognitive Feedback in an Intelligent Tutoring System". In: *Learning and Instruction* 21.2, S. 267–280. ISSN: 09594752. DOI: 10.1016/j.learninstruc.2010.07.004.

Rüdian, Sylvio et al. (März 2025). „Feedback on Feedback: Student's Perceptions for Feedback from Teachers and Few-Shot LLMs". In: *Proceedings of the 15th International Learning Analytics and Knowledge Conference*. Dublin Ireland: ACM, S. 82–92. ISBN: 979-8-4007-0701-8. DOI: 10.1145/3706468.3706479.

Ruf, Alexander, Marc Berges und Peter Hubwieser (2015). „Classification of Programming Tasks According to Required Skills and Knowledge Representation". In: *Informatics in Schools. Curricula, Competences, and Competitions*. Hrsg. von Andrej Brodnik und Jan Vahrenhold. Bd. 9378. Cham: Springer International Publishing, S. 57–68. ISBN: 978-3-319-25395-4 978-3-319-25396-1. DOI: 10.1007/978-3-319-25396-1_6.

Russell, Stuart J. und Peter Norvig (2023). *Künstliche Intelligenz: ein moderner Ansatz*. 4., aktualisierte Auflage. München: Pearson. ISBN: 978-3-86894-430-3 978-3-86326-326-3.

Sadler, D. Royce (Juni 1989). „Formative Assessment and the Design of Instructional Systems". In: *Instructional Science* 18.2, S. 119–144. ISSN: 0020-4277, 1573-1952. DOI: 10.1007/BF00117714.

Saghiri, Ali Mohammad et al. (Apr. 2022). „A Survey of Artificial Intelligence Challenges: Analyzing the Definitions, Relationships, and Evolutions". In: *Applied Sciences* 12.8, S. 4054. ISSN: 2076-3417. DOI: 10.3390/app12084054.

Sales, G.C. (1993). „Adapted and Adaptive Feedback in Technology-Based Instruction". In: *Interactive Instruction and Feedback*. Englewood Cliffs, N.J, S. 159–175. JSTOR: 44428189.

Sampaio De Alencar, Rafaella et al. (Juni 2025). „Integrating Expert Knowledge With Automated Knowledge Component Extraction for Student Modeling". In: *Proceedings of the 33rd ACM Conference on User Modeling, Adaptation and Personalization*. New York City USA: ACM, S. 307–312. ISBN: 979-8-4007-1313-2. DOI: 10.1145/3699682.3728348.

Sarsa, Sami et al. (Aug. 2022). „Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models". In: *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1*. Lugano und Virtual Event Switzerland: ACM, S. 27–43. ISBN: 978-1-4503-9194-8. DOI: 10.1145/3501385.3543957.

Savelka, Jaromir et al. (2023). „Large Language Models (GPT) Struggle to Answer Multiple-Choice Questions About Code:" In: *Proceedings of the 15th International Conference on*

Computer Supported Education. Prague, Czech Republic: SCITEPRESS - Science and Technology Publications, S. 47–58. ISBN: 978-989-758-641-5. DOI: 10.5220/0011996900003470.

Schabram, Katharina (2007). „Lernaufgaben im Unterricht. Instruktionspsychologische Analysen am Beispiel der Physik." Diss. Duisburg, Essen: Universität Duisburg-Essen.

Schlüter, Ute (Apr. 2012). „Implementierung einer Qualitätsstrategie im Feedbackmanagement". In: Hrsg. von Konrad Umlauf. DOI: 10.18452/2066.

Scholl, Andreas, Daniel Schiffner und Natalie Kiesler (2024). „Analyzing Chat Protocols of Novice Programmers Solving Introductory Programming Tasks with ChatGPT". In: *Proceedings of DELFI 2024*. DOI: 10.18420/DELFI2024_05.

Schreier, Margrit (2012). *Qualitative Content Analysis in Practice*. 1 Oliver's Yard, 55 City Road London EC1Y 1SP: SAGE Publications Ltd. ISBN: 978-1-84920-593-1 978-1-5296-8257-1. DOI: 10.4135/9781529682571.

Shavelson, Richard J. und Paula Stern (Dez. 1981). „Research on Teachers' Pedagogical Thoughts, Judgments, Decisions, and Behavior". In: *Review of Educational Research* 51.4, S. 455–498. ISSN: 0034-6543, 1935-1046. DOI: 10.3102/00346543051004455.

Shute, Valerie J. (März 2008). „Focus on Formative Feedback". In: *Review of Educational Research* 78.1, S. 153–189. ISSN: 0034-6543, 1935-1046. DOI: 10.3102/0034654307313795.

Simon, Herbert Alexander (2008). *The Sciences of the Artificial*. 3. ed., [Nachdr.] Cambridge, Mass.: MIT Press. ISBN: 978-0-262-19374-0 978-0-262-69191-8.

Singer, Gadi et al. (2023). „Thrill-K Architecture: Towards a Solution to the Problem of Knowledge Based Understanding". In: *Artificial General Intelligence*. Hrsg. von Ben Goertzel et al. Bd. 13539. Cham: Springer International Publishing, S. 404–412. ISBN: 978-3-031-19906-6 978-3-031-19907-3. DOI: 10.1007/978-3-031-19907-3_39.

Singla, Alex et al. (März 2025). *The State of AI: How Organizations Are Rewiring to Capture Value*.

Sleeman, D. und John Seely Brown, Hrsg. (1982). *Intelligent Tutoring Systems*. Computers and People Series. London ; New York: Academic Press. ISBN: 978-0-12-648680-3.

Smith, Stanley G. und Bruce Arne Sherwood (Apr. 1976). „Educational Uses of the PLATO Computer System: The PLATO System Is Used for Instruction, Scientific Research, and Communications." In: *Science* 192.4237, S. 344–352. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.769165.

Stokes, Donald E. (1997). *Pasteur's Quadrant: Basic Science and Technological Innovation*. Washington, D.C: Brookings Institution Press. ISBN: 978-0-8157-8178-3 978-0-8157-8177-6.

Stovner, Roar Bakken und Kirsti Klette (Feb. 2022). „Teacher Feedback on Procedural Skills, Conceptual Understanding, and Mathematical Practices: A Video Study in Lower Secondary Mathematics Classrooms". In: *Teaching and Teacher Education* 110, S. 103593. ISSN: 0742051X. DOI: 10.1016/j.tate.2021.103593.

Strickroth, Sven (Juli 2024). „Scalable Feedback for Student Live Coding in Large Courses Using Automatic Error Grouping". In: *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. Milan Italy: ACM, S. 499–505. ISBN: 979-8-4007-0600-4. DOI: 10.1145/3649217.3653620.

Strickroth, Sven, Hannes Olivier und Niels Pinkwart (2011). „Das GATE-system: Qualitätssteigerung Durch Selbsttests Für Studenten Bei Der Onlineabgabe von Übungsaufgaben?" In: *DeLFI 2011 - Die 9. e-Learning Fachtagung Informatik*. Bonn: Gesellschaft für Informatik e.V., S. 115–126. ISBN: 978-3-88579-282-6.

Strickroth, Sven und Michael Striewe (Nov. 2022). „Building a Corpus of Task-Based Grading and Feedback Systems for Learning and Teaching Programming". In: *International Journal of Engineering Pedagogy (iJEP)* 12.5, S. 26–41. ISSN: 2192-4880. DOI: 10.3991/ijep.v12i5.31283.

Striewe, Michael (Nov. 2016). „An Architecture for Modular Grading and Feedback Generation for Complex Exercises". In: *Science of Computer Programming* 129, S. 35–47. ISSN: 01676423. DOI: 10.1016/j.scico.2016.02.009.

Südkamp, Anna und Anna-Katharina Praetorius, Hrsg. (2017). *Diagnostische Kompetenz von Lehrkräften: theoretische und methodische Weiterentwicklungen*. Pädagogische Psychologie und Entwicklungspsychologie 94. Münster New York: Waxmann. ISBN: 978-3-8309-3596-4.

Sweller, John (Apr. 1988). „Cognitive Load During Problem Solving: Effects on Learning". In: *Cognitive Science* 12.2, S. 257–285. ISSN: 0364-0213, 1551-6709. DOI: 10.1207/s15516709cog1202_4.

Sweller, John, Jeroen J. G. Van Merrienboer und Fred G. W. C. Paas (1998). „Cognitive Architecture and Instructional Design". In: *Educational Psychology Review* 10.3, S. 251–296. ISSN: 1040726X. DOI: 10.1023/A:1022193728205.

Tahsin, Noshin, Nafis Fuad und Abdus Satter (Feb. 2023). *Prevalence of 'Atoms of Confusion' in Open Source Java Systems: An Empirical Study*. DOI: 10.22541/au.167570695.54470176/v1.

Taylor, Andrew et al. (März 2024). „Dcc –Help: Transforming the Role of the Compiler by Generating Context-Aware Error Explanations with Large Language Models". In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. Portland OR USA: ACM, S. 1314–1320. ISBN: 979-8-4007-0423-9. DOI: 10.1145/3626252.3630822.

Thangaraj, Jagadeeswaran, Monica Ward und Fiona O'Riordan (2024). „An Experience with Adaptive Formative Assessment for Motivating Novices in Introductory Programming Learning". In: *OASIcs, Volume 122, ICPEC 2024* 122. Hrsg. von André L. Santos und Maria Pinto-Albuquerque, 6:1–6:12. ISSN: 2190-6807. DOI: 10.4230/OASICS.ICPEC.2024.6.

UNESCO (2025). *Guidance for Generative AI in Education and Research | UNESCO*. URL: https://www.unesco.org/en/articles/guidance-generative-ai-education-and-research.

Uschold, Michael und Michael Grüninger (Jan. 1996). „Ontologies: Principles, Methods and Applications". In: *The Knowledge Engineering Review* 11.

Van Bekkum, Michael et al. (Sep. 2021). „Modular Design Patterns for Hybrid Learning and Reasoning Systems: A Taxonomy, Patterns and Use Cases". In: *Applied Intelligence* 51.9, S. 6528–6546. ISSN: 0924-669X, 1573-7497. DOI: 10.1007/s10489-021-02394-3.

Vanlehn, Kurt (Aug. 2006). „The Behavior of Tutoring Systems". In: 16.3, S. 227–265. ISSN: 1560-4292.

Vanlehn, Kurt (Okt. 2011). „The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems". In: *Educational Psychologist* 46.4, S. 197–221. ISSN: 0046-1520, 1532-6985. DOI: 10.1080/00461520.2011.611369.

Vaswani, Ashish et al. (2017). *Attention Is All You Need*. DOI: 10.48550/ARXIV.1706.03762.

Vee, Annette (2017). *Coding Literacy: How Computer Programming Is Changing Writing*. Software Studies. Cambridge, Massachusetts London, England: The MIT Press. ISBN: 978-0-262-03624-5 978-0-262-34023-6.

Von Rueden, Laura et al. (2021). „Informed Machine Learning - A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems". In: *IEEE Transactions on Knowledge and Data Engineering*, S. 1–1. ISSN: 1041-4347, 1558-2191, 2326-3865. DOI: 10.1109/TKDE.2021.3079836.

Vygotsky, L. S. (Okt. 1980). *Mind in Society: Development of Higher Psychological Processes*. Hrsg. von Michael Cole et al. Harvard University Press. ISBN: 978-0-674-07668-6 978-0-674-57628-5. DOI: 10.2307/j.ctvjf9vz4. JSTOR: 10.2307/j.ctvjf9vz4.

Wagner, Salome et al. (Feb. 2024). „The More, the Better? Learning with Feedback and Instruction". In: *Learning and Instruction* 89, S. 101844. ISSN: 09594752. DOI: 10.1016/j.learninstruc.2023.101844.

Wand, Y. und R. Weber (Okt. 1993). „On the Ontological Expressiveness of Information Systems Analysis and Design Grammars". In: *Information Systems Journal* 3.4, S. 217–237. ISSN: 1350-1917, 1365-2575. DOI: 10.1111/j.1365-2575.1993.tb00127.x.

Wang, Sierra, John Mitchell und Chris Piech (März 2024). „A Large Scale RCT on Effective Error Messages in CS1". In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. Portland OR USA: ACM, S. 1395–1401. ISBN: 979-8-4007-0423-9. DOI: 10.1145/3626252.3630764.

Wang, Zhen et al. (Juli 2019). „Elaborated Feedback and Learning: Examining Cognitive and Motivational Influences". In: *Computers & Education* 136, S. 130–140. ISSN: 03601315. DOI: 10.1016/j.compedu.2019.04.003.

Watson, Christopher und Frederick W.B. Li (2014). „Failure Rates in Introductory Programming Revisited". In: *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education - ITiCSE '14*. Uppsala, Sweden: ACM Press, S. 39–44. ISBN: 978-1-4503-2833-3. DOI: 10.1145/2591708.2591749.

Weich, Andreas et al. (Okt. 2021). „Adaptive Lernsysteme Zwischen Optimierung Und Kritik: Eine Analyse Der Medienkonstellationen Bettermarks Aus Informatischer Und Medienwissenschaftlicher Perspektive". In: *MedienPädagogik: Zeitschrift für Theorie und Praxis der Medienbildung* 44, S. 22–51. ISSN: 1424-3636. DOI: 10.21240/mpaed/44/2021.10.27.X.

Weinert, Franz Emanuel (2001). „Concept of Competence: A Conceptual Clarification". In: *Defining and Selecting Key Competencies*. Hrsg. von Dominique Simone Rychen und Laura Hersh Salganik. Seattle: Hogrefe & Huber, S. 45–65. ISBN: 0-88937-248-9.

Wermelinger, Michel (März 2023). „Using GitHub Copilot to Solve Simple Programming Problems". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. Toronto ON Canada: ACM, S. 172–178. ISBN: 978-1-4503-9431-4. DOI: 10.1145/3545945.3569830.

Westervelt, Eric R. et al. (2007). *Feedback Control of Dynamic Bipedal Robot Locomotion*. Control and Automation 1. Boca Raton London New York: CRC Press, Taylor & Francis Group. ISBN: 978-1-4200-5372-2.

Wichmann, Astrid et al. (Jan. 2014). „Making the Most out of It: Maximizing Learners' Benefits from Expert, Peer and Automated Feedback across Domains". In: *Proceedings of International Conference of the Learning Sciences, ICLS*. Bd. 3, S. 1416–1425.

Wiener, Norbert (Okt. 2019). *Cybernetics or Control and Communication in the Animal and the Machine*. The MIT Press. ISBN: 978-0-262-35590-2. DOI: 10.7551/mitpress/11810.001.0001.

Wiggins, Grant (Apr. 2011). „A True Test: Toward More Authentic and Equitable Assessment". In: *Phi Delta Kappan* 92.7, S. 81–93. ISSN: 0031-7217, 1940-6487. DOI: 10.1177/003172171109200721.

Wilson, Brent G., Hrsg. (1996). *Constructivist Learning Environments: Case Studies in Instructional Design*. Englewood Cliffs, N.J: Educational Technology Publications. ISBN: 978-0-87778-290-2.

Wisniewski, Benedikt, Klaus Zierer und John Hattie (Jan. 2020). „The Power of Feedback Revisited: A Meta-Analysis of Educational Feedback Research". In: *Frontiers in Psychology* 10, S. 3087. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2019.03087.

Wolf, Kenneth und Ellen A. Stevens (2007). „The Role of Rubrics in Advancing and Assessing Student Learning". In: *The Journal of Effective Teaching* 7, S. 3–14.

Xiao, Ruiwei, Xinying Hou und John Stamper (Mai 2024). „Exploring How Multiple Levels of GPT-Generated Programming Hints Support or Disappoint Novices". In: *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. Honolulu HI USA: ACM, S. 1–10. ISBN: 979-8-4007-0331-7. DOI: 10.1145/3613905.3650937.

Yan, Yi-Miao et al. (Feb. 2025). „LLM-based Collaborative Programming: Impact on Students' Computational Thinking and Self-Efficacy". In: *Humanities and Social Sciences Communications* 12.1, S. 149. ISSN: 2662-9992. DOI: 10.1057/s41599-025-04471-1.

Yazdani, Shamim et al. (Okt. 2025). „Generative AI in Depth: A Survey of Recent Advances, Model Variants, and Real-World Applications". In: *Journal of Big Data* 12.1, S. 230. ISSN: 2196-1115. DOI: 10.1186/s40537-025-01247-x.

Yilmaz, Ramazan und Fatma Gizem Karaoglan Yilmaz (2023). „The Effect of Generative Artificial Intelligence (AI)-Based Tool Use on Students' Computational Thinking Skills, Programming Self-Efficacy and Motivation". In: *Computers and Education: Artificial Intelligence* 4, S. 100147. ISSN: 2666920X. DOI: 10.1016/j.caeai.2023.100147.

Zamfirescu-Pereira, J.D. et al. (Apr. 2023). „Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts". In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. Hamburg Germany: ACM, S. 1–21. ISBN: 978-1-4503-9421-5. DOI: 10.1145/3544548.3581388.

Zehetmeier, Daniela et al. (Juni 2015). „Development of a Classification Scheme for Errors Observed in the Process of Computer Programming Education". In: *HEAd'15. Conference on Higher Education Advances*. Hrsg. von Editorial Universitat Politècnica de València. Editorial Universitat Politècnica de València, S. 475–484. ISBN: 978-84-9048-340-4. DOI: 10.4995/HEAd15.2015.356.

Zhang, Jialu et al. (Apr. 2024). „PyDex: Repairing Bugs in Introductory Python Assignments Using LLMs". In: *Proceedings of the ACM on Programming Languages* 8.OOPSLA1, S. 1100–1124. ISSN: 2475-1421. DOI: 10.1145/3649850.

Liste eigener Publikationen

Berges, Marc et al. (2023). „Learning Support Systems Based on Mathematical Knowledge Management". In: *Intelligent Computer Mathematics*. Hrsg. von Catherine Dubois und Manfred Kerber. Bd. 14101. Cham: Springer Nature Switzerland, S. 84–97. ISBN: 978-3-031-42752-7 978-3-031-42753-4. DOI: 10.1007/978-3-031-42753-4_6.

Betzendahl, Jonas et al. (2025). „Efficient Exam Correction at Scale - Streamlining Paper-Based Assessments with the VoLl-KOrN System". In: *Digitales Lehren und Lernen an der Hochschule: Strategien - Bedingungen - Umsetzung*. 1. Auflage. Hochschulbildung: Lehre und Forschung 9. Bielefeld: transcript. ISBN: 978-3-8394-7120-3 978-3-8376-7120-9. DOI: 10.14361/9783839471203-027.

Grelka, Felix, Dominic Lohr und Marc Berges (2025). „ALeA : Advancing Personalized Learning with Adaptive Assistance and Semantic Annotation". In: *Workshop KI Und Bildung 2024*, S. 1–2. DOI: 10.20378/irb-108296.

Jeuring, Johan et al. (Juli 2022a). „Steps Learners Take When Solving Programming Tasks, and How Learning Environments (Should) Respond to Them". In: *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 2*. Dublin Ireland: ACM, S. 570–571. ISBN: 978-1-4503-9200-6. DOI: 10.1145/3502717.3532168.

Jeuring, Johan et al. (Dez. 2022b). „Towards Giving Timely Formative Feedback and Hints to Novice Programmers". In: *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education*. Dublin Ireland: ACM, S. 95–115. ISBN: 979-8-4007-0010-1. DOI: 10.1145/3571785.3574124.

Kiesler, Natalie, Dominic Lohr und Hieke Keuning (Okt. 2023). „Exploring the Potential of Large Language Models to Generate Formative Programming Feedback". In: *2023 IEEE Frontiers in Education Conference (FIE)*. College Station, TX, USA: IEEE, S. 1–5. ISBN: 979-8-3503-3642-9. DOI: 10.1109/FIE58773.2023.10343457.

Kruse, Theresa et al. (2023). „Learning with ALeA: Tailored Experiences through Annotated Course Material". In: *INFORMATIK 2023 - Designing Futures: Zukünfte Gestalten*. Bonn: Gesellschaft für Informatik e.V., S. 395–398. ISBN: 978-3-88579-731-9. DOI: 10.18420/INF2023_43.

Kruse, Theresa et al. (2024). „Term Extraction for Domain Modeling". In: *Proceedings of DELFI 2024*. DOI: 10.18420/DELFI2024_33.

Lindner, Annabel et al. (2023). „Von Autonomem Fahren bis Zahnarzt - Vorstellungen von Schüler:innen zu Künstlicher Intelligenz und ihre Integration in den Informatikunterricht". In: *INFOS 2023 - Informatikunterricht zwischen Aktualität und Zeitlosigkeit*. Gesellschaft für Informatik e.V., S. 93–102. ISBN: 978-3-88579-730-2. DOI: 10.18420/INFOS2023-007.

Lohr, Dominic und Marc Berges (Sep. 2021). „Towards Criteria for Valuable Automatic Feedback in Large Programming Classes". In: *Hochschuldidaktik Informatik HDI 2021*. Dortmund: Jörg Desel, Simone Opel, S. 181–186. ISBN: 978-3-00-070267-9.

Lohr, Dominic und Marc Berges (2025). „Der Weg ist das Ziel – Ein skalierbarer Wechsel von summativem zu formativem Assessment in der Programmierausbildung mit Antwortklassen". In: *Proceedings Seventh Workshop "Automatische Bewertung von Programmieraufgaben"(ABP 2025)*. Gesellschaft für Informatik. DOI: 10.18420/ABP2025_05.

Lohr, Dominic, Hieke Keuning und Natalie Kiesler (Feb. 2025). „You're (Not) My Type – Can LLMs Generate Feedback of Specific Types for Introductory Programming Tasks?" In: *Journal of Computer Assisted Learning* 41.1, e13107. ISSN: 0266-4909, 1365-2729. DOI: 10.1111/jcal.13107.

Lohr, Dominic, Lars Wechsler und Marc Berges (Juni 2025). „Introducing a Self-Study Course for Learning Textual Programming at Highschool with APFEL and ALeA". In: *Workshop KI Und Bildung 2024*. DOI: 10.20378/irb-108298.

Lohr, Dominic et al. (2023a). „The Potential of Answer Classes in Large-scale Written Computer-Science Exams". In: *Hochschuldidaktik Informatik HDI 2023*. Dortmund: Jörg Desel, Simone Opel, S. 179–189. ISBN: 978-3-00-076206-2.

Lohr, Dominic et al. (Aug. 2023b). „The Y-Model - Formalization of Computer Science Tasks in the Context of Adaptive Learning Systems". In: *2023 IEEE 2nd German Education Conference (GECon)*. Berlin, Germany: IEEE, S. 1–6. ISBN: 979-8-3503-4813-2. DOI: 10.1109/GECon58119.2023.10295148.

Lohr, Dominic et al. (Juli 2024a). „"Let Them Try to Figure It Out First Reasons Why Experts (Do Not) Provide Feedback to Novice Programmers". In: *Proceedings of the 2024 Innovation and Technology in Computer Science Education (ITiCSE 2024)*. Bd. 1. Milan, Italy: ACM, S. 7. DOI: 10.1145/3649217.3653530.

Lohr, Dominic et al. (2024b). „Adaptive Learning Systems in Programming Education: A Prototype for Enhanced Formative Feedback". In: *Proceedings of DELFI 2024*. Fulda: Gesellschaft für Informatik e.V., S. 549–554. DOI: 10.18420/DELFI2024_57.

Lohr, Dominic et al. (Okt. 2025a). „"Ignore These Errors for Now- How Experts Provide Feedback on Steps Novices Take Towards Solving Programming Problems". In: *Proceedings of the ACM Global on Computing Education Conference 2025 Vol 1*. Gaborone Botswana: ACM, S. 149–155. ISBN: 979-8-4007-1929-5. DOI: 10.1145/3736181.3747150.

Lohr, Dominic et al. (Feb. 2025b). „Leveraging Large Language Models to Generate Course-Specific Semantically Annotated Learning Objects". In: *Journal of Computer Assisted Learning* 41.1, e13101. ISSN: 0266-4909, 1365-2729. DOI: 10.1111/jcal.13101.

Lohr, Dominic et al. (2025c). „The Potential of Answer Classes in Large-scale Written Computer-Science Exams – Vol.2". In: *Commentarii informaticae didacticae*. Aachen: Jörg Desel, Simone Opel. DOI: 10.25932/publishup-68780.

Tabellenverzeichnis

Abbildungsverzeichnis

Anhang A

Personas

Tabelle A.1: Originaltexte der Persona im Fragebogen

Persona	Sequenz-ID	Kurzbeschreibung der Persona
 BOB	B02 CodingBat Java	*This is **Bob**, 31, a novice learner of programming. He is enrolled in a computing degree program at a German University of Applied Sciences. He successfully passed his first programming course and thinks of programming as a tool to solve problems, and to automate his home in the long term. Bob considers control structures and loops as easy. Tasks involving classes and objects, or recursion are of medium difficulty for him. He thinks his theoretical knowledge and programming competencies are good, so he does not hesitate to participate in a study on learning environments for programming in a usability laboratory.*
 ALICE	B05 CodingBat Java	*Meet **Alice**, 19, a computer science student at a German University of Applied Sciences. She just completed her first programming course and passed it with a good grade. She thinks her programming knowledge and competencies are good. Some tasks are easy for her, like the ones with objects and classes, or arrays. She also perceives recursion as difficult. Alice wants to learn programming to develop her own apps and software once she graduates. During her first semester break, she hears about a usability experiment at her university where one can do some programming. She is curious and decides to participate to see that usability laboratory from the inside.*
 EVE	148 FITech Dart	***Eve** is a 30-year-old marketing specialist, who embraces the "FITech 101 Introduction to Programming" course to conquer programming basics. Eve aims to grasp programming basics and Dart language fundamentals to better communicate with her technical team and gain a foundational understanding of coding logic. Her goal is not to become a programmer per se, but to confidently write simple scripts and comprehend coding concepts. New to programming, Eve is apprehensive about the learning curve. Balancing her job and other commitments with the course also concerns her. Nonetheless, she embraces the course as an opportunity for personal growth.*

Persona	Sequenz-ID	Kurzbeschreibung der Persona
 DAVE	538 FITech Dart	*Meet **Dave**, a 33-year-old sales manager intrigued by the world of programming, despite his limited exposure to Excel macros. Dave's role involves analyzing sales data, and he believes that learning programming through FITech 101 Introduction to Programming could amplify his analytical capabilities. Dave's primary goal is to understand the fundamentals of programming and gain proficiency in Dart to automate tasks and enhance his data analysis skills. He envisions using code to manipulate and interpret sales data more effectively. With little programming background, Dave anticipates challenges in grasping new concepts. Finding time for the course amidst his managerial responsibilities is another concern. However, Dave's success in the first weeks encouraged him to explore more advanced programming techniques to further optimize his sales analysis processes.*
 PARKER	tt1 TaskTracker Python	*This is **Parker**, a 16-year old high school student. A few months ago, Parker started a new hobby: programming. Parker's mother is a software engineer, and introduced them to the Python programming language. They already wrote a few simple scripts. Parker decided to join some programming classes offered at school, because they are thinking about studying Computer Science after high school. In this class, the students have to work with a professional IDE. Parker is very intrigued by this tool, but struggles a bit with all of its options.*
 JASMINE	tt2 TaskTracker Python	***Jasmine** is a 14-year old high school student. She enjoys school very much. Because most classes are not so hard for her, she follows extra classes. Recently, she has taken up programming. At the moment, she wants to pursue a career in biology, but she thinks programming will be a useful skill to have for that. She has learned the basics of the Python syntax, but is struggling with using the right programming constructs to solve problems.*