“Ignore These Errors for Now” – How Experts Provide Feedback on Steps Novices Take Towards Solving Programming Problems

Dominic Lohr
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
dominic.lohr@fau.de

Natalie Kiesler
Nuremberg Tech
Nuremberg, Germany
natalie.kiesler@th-nuernberg.de

Hieke Keuning
Utrecht University
Utrecht, The Netherlands
h.w.keuning@uu.nl

Johan Jeuring
Utrecht University
Utrecht, The Netherlands
j.t.jeuring@uu.nl

Abstract

Formative feedback can be crucial for developing programming competencies, especially among novice learners. Despite the importance of appropriate timing and feedback content, there is often no consensus among experts *when* and *how* to give feedback on steps learners take when solving programming tasks. This disagreement is due to the subjective nature of expert assessments, which often rely on personal experience and intuition. To better understand how experts give feedback, we surveyed a diverse group of programming educators on *how* to give feedback on steps of a novice's programming process. We qualitatively analyzed the open-ended answers and developed a classification representing the categories of human feedback in the context of programming education. The results show that the existing programming feedback taxonomies cannot be applied 1:1 to expert feedback as they were designed for automated systems and cannot adequately cover the nuances of human expert interventions. Thereby, this work has implications for educators aiming to improve instruction and feedback strategies, but also researchers studying (human) feedback and its effects.

CCS Concepts

• **Social and professional topics → Computer science education**.

Keywords

Learning programming, novice programmers, formative feedback

ACM Reference Format:
Dominic Lohr, Natalie Kiesler, Hieke Keuning, and Johan Jeuring. 2025. “Ignore These Errors for Now” – How Experts Provide Feedback on Steps Novices Take Towards Solving Programming Problems. In *Proceedings of the ACM Global Computing Education Conference 2025 Vol 1 (CompEd 2025), October 21–25, 2025, Gaborone, Botswana.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3736181.3747150

CompEd 2025, Gaborone, Botswana

ACM ISBN 979-8-4007-1929-5/2025/10
https://doi.org/10.1145/3736181.3747150

1 Introduction

Learning to program is challenging and involves a multitude of missteps and errors [17, 23]. Even experienced developers are confronted with faulty code on a daily basis – but unlike novices, they have a "Swiss Army knife" full of tools and experience to debug their code. Especially when beginning to learn programming, support plays a fundamental role.

Giving individual feedback to programming novices is often infeasible for educators with hundreds of students in their classes. The feedback that automated systems provide is often simple, and focus on symptoms of errors instead of addressing the causes of errors or providing hints on how to proceed – the kinds of feedback learners consider more valuable [14]. Intelligent tutoring systems (ITSs) designed to replicate the effectiveness of one to one human tutoring might be a potential solution to this resource and feedback quality problem. Developers of ITSs for programming education (Intelligent Programming Tutors [3]) try to replicate human tutor behavior to provide learners with immediate and adaptive feedback. They need answers to the questions: (1) *why* does an expert give feedback in a particular situation, and (2) *how*? Previous research investigated experts' reasons for providing or withholding feedback on steps students take when solving introductory programming problems [16]. We are not aware of work within programming education on the second question. To provide more valuable feedback to learners who work on programming tasks we therefore need a deeper understanding of the strategies and behaviors of experienced programming educators, i.e., experts.

Our study is led by the following **research question**: *How do experts provide feedback to novice programmers who are solving programming tasks?*

The **main contribution** of this research is the development of a set of categories that describe the types of feedback experts give to programming novices. These categories can be used in future research to analyze expert feedback strategies, evaluate the quality of feedback, and determine how well feedback meets the informational needs of learners. Furthermore, our findings can be used to compare expert feedback with feedback from peers, educational tools or AI models, hereby identifying differences and informing improvements in feedback strategies.

2 Related Work

Feedback in educational contexts has been studied extensively, and is considered a major factor for successful learning [7, 8, 12, 20]. Feedback encompasses multiple facets, such as function or objective, presentation, and content. These facets influence whether students perceive it as valuable or not [20]. In this study, we focus on the *content* of the feedback message, wich can be categorized into two main components: verification (evaluative), and elaboration (informational) [13]. Narciss presents these in a "content-related classification of feedback" as simple and elaborated components. Simple components are binary indications of correctness, a percentage, or the correct answer itself. The elaborated components are organized with respect to the aspect of the instructional context that is addressed: the task itself, conceptual knowledge, mistakes, procedural knowledge, and metacognitive knowledge.

Hattie and Timperley's model [8] states that effective feedback answers the questions "where am I going", "how am I going", and "where to next". These questions operate at four levels: (1) *task*, (2) *process*, (3) *self-regulation* and (4) *self*. Ott et al. [21] explored the applicability of Hattie and Timperley's model in the CS1 context and determined that automated feedback contributes to the first three levels. They excluded the *self* level (4), as it lacks a significant impact on enhancing learning performance [21]. In addition, Ott et al. devised a roadmap from Hattie and Timperleys' model which outlines effective feedback practices for various levels and stages.

D'Mello et al. examined the feedback strategies of 10 expert tutors over 50 sessions in different domains like algebra, geometry, physics, chemistry, and biology [5]. They investigated whether the feedback was direct and immediate, or indirect and delayed, as well as if experts' feedback strategies were modulated by domain. Their results show that expert tutors' feedback is direct, immediate, discriminating, and largely domain independent.

There are several studies that investigate feedback in the context of learning programming. Jeuring et al. [9] and Rocha et al. [24] both emphasize the importance of timely, formative feedback. A recent ITiCSE working group [9] focused on how experts would give feedback to students solving programming tasks. They studied *when* students need feedback and hints, and *how* this feedback should be given. Their results comprise guidelines that identify *when* and *how* to intervene, describing events such as trial and error behavior, logical errors, or subgoal completion as potential points of intervention for educators. Rocha et al. advanced this by proposing a framework for adaptive feedback, which can be used by teaching assistants to guide their pedagogical decision-making [24].

Dong et al. [6] investigated tutor interventions during block-based programming assignments. They found key reasons for intervening, including *missing components to make progress*, *using wrong or unnecessary blocks*, *misusing needed blocks*, *critical logic errors*, *needing confirmation and next steps*, and *unclear student intention.*

Cucuiat and Waite [4] conducted a pilot study in which secondary school instructors assessed LLM-generated feedback messages on programming errors. They found that teachers favor the role of feedback as a "guiding and developing understanding role", rather than a "telling role" (e.g. directly providing the solution). This role complemented existing guidelines on programming error messages [2], also in line with the teachers' preferences.

Keuning et al. [10] applied and extended Narciss' categorization [20] for the programming domain, refining and adding new subcategories to classify feedback content in more detail. This classification was aimed at automated feedback, and applied to and tested with a large number of digital learning tools that provide such feedback. It has not been applied to human feedback yet, and is the starting point for the coding we describe in Section 3.

To conclude, there is an extensive body on feedback research, also in the context of (introductory) programming education. Some studies go beyond providing feedback and look more closely at its content and timing. Yet, we have found little research on how experts give feedback at certain steps or for specific concerns while learning how to program.

3 Methodology

To answer our research question, we (1) selected a number of sequences consisting of steps students take when solving a programming task, (2) presented these to experts through a survey, and (3) asked them to annotate these steps with feedback. We then analyzed the data to categorize the expert feedback. We define experts as people with years of teaching experience in programming classes.

3.1 Dataset Selection

To gather experts' feedback statements, we used available (published) datasets containing students' steps while solving introductory programming tasks. The original datasets were collected through online learning environments, and published by the 2022 ITiCSE WG [9]. In previous work [16], we created six sequences of students' steps from this dataset, which we displayed to experts. The selected sequences[1] cover three different tasks in three different programming languages, see Table 1. Some sequences include all steps a student takes to solve a problem, while others comprise only parts of (longer) sequences. All sequences comprise interesting steps (e.g., with student errors), and satisfy the following criteria:

- Task is designed for novice learners of programming.
- Length of model solution ranges between 10 and 15 lines of code.
- Sequences contain errors where feedback would be helpful.
- Successful completion of the task is not required.
- Sequences in which a student starts with the problem-solving process are preferred.

The steps in these sequences sometimes combine several steps in the underlying sequences, because the granularity in the original sequences is too fine-grained to show them to teachers. The resulting steps consist of:

- Initial code provided by the learning environment, if present.
- Code execution and/or feedback request, e.g. via a compile, run, or hint-button.
- Pasting in code.
- Step before and after the pause, for pauses of 2 minutes or longer.
- Every finished line of code including open and closing parentheses that might appear on the previous or next line, when a student continuously writes code.

[1]The sequences can be accessed on https://github.com/Programming-Steps-Working-Group-2022/Expert-Reasons-ITiCSE-24

Table 1: Overview of data sequences.

Exercise	Lang.	Description	Source	Concepts	Name	Id	Steps
Fibonacci	Java	Compute the n-th Fibonacci number using recursion.	CodingBat	methods, conditionals, recursion	Bob	B02	9
					Alice	B05	10
Password	Dart	Write a function asking the user for a password until they enter the correct one (given as parameter).	FITech	methods, parameters, loops, conditionals, input/output	Eve	148	23
					Dave	538	15
MaxDigit	Python	Given a string containing only digits, find and print the largest digit.	TaskTracker	loops, conditionals, input/output	Parker	tt1	12
					Jasmine	tt2	22

- First instance when a student makes an error (e.g., a typo) within a keyword.
- Step before and after the formatting in case of a formatting error.

In the resulting sequences the majority of the steps add or change (part of) a line in a program. See Fig. 1 for an example of a step where a student completed a line and pressed the Run-button.

3.2 Survey Design

The survey asked the participants some general questions, e.g., job title, years of experience teaching programming, school level, main country of teaching, and in which programming languages they feel comfortable (C# and Java, or Python, or all). In our dataset, Dart is almost indistinguishable from C# and Java. Since Dart is much less well known, we assumed that people feeling comfortable giving feedback on C# and Java programs would be fine with giving feedback on beginners' Dart programs as well.

After the general questions, we randomly offered one of the six sequences to the participants, taking into account their answer to the programming languages in which they feel comfortable giving feedback. Each of the series of questions on sequences of steps starts with the description of the task, including some test cases. We then described a student persona solving the task. We developed six personas based on our knowledge of the contexts and target group in which the datasets had been collected. For the questions related to the expert feedback, we gave the following instructions:

> *In the next couple of questions we will ask what kind of feedback you would give to Bob. 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. [...] 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.*

Fig. 1 gives an example of a step that was shown to the participants. We always presented the preceding step to indicate the students' progress. For each step, we asked the participants the following three questions (1) Would you give feedback and/or hints at this point? (Yes/Maybe/No), (2) Why? (open question), and (3) Which feedback and/or hints? (open question). In this study, we evaluate the responses to the third question.

3.3 Data Gathering from Experts

Before distributing our survey, our institutional ethics committee approved our research proposal. All participants gave consent to use their data for our research.

3.3.1 Distribution of the Survey. The survey was distributed in several ways: on the SIGSCE mailing list, at talks at conferences and universities (ITiCSE 2023, DELFI 2023, FIE 2023, several regional secondary school CS teachers meetings, a national secondary school CS teachers meeting, Università della Svizzera italiana), and via snowballing to (international) colleagues with several years of teaching experience in programming classes.

3.3.2 Description of the Sample. We received 47 responses; mostly from Europe (26, with 9 from Germany, and 11 from the Netherlands) and North-America (14, with 11 from the US), but we also had respondents from Middle America, South America and Africa. Our respondents had on average 13.28 years of teaching experience (median 8), with quite a spread: the standard deviation was 11.65. We did not gather any other demographic data.

Respondents were distributed over various sequences (within parentheses is the number of answers to the *how* question for each sequence): Bob 6 (40), Alice 9 (33), Eve 10 (42), Dave 5 (25), Parker 13 (45), and Jasmine 4 (25). In total, we analyzed 210 feedback messages. We did not enforce answering the feedback questions, and some respondents answered only part of the feedback questions. We included the questions they answered in our analysis.

3.4 Data Analysis

For the analysis of experts' responses to open-ended questions about *how* they provided feedback, we applied a qualitative analysis [19]. We coded all feedback messages by the experts using Keuning et al.'s [10] categorization of automatically generated feedback as a starting point. All but the last two categories of Table 3 come from this categorization. In general, each expert feedback to a student step was treated as a coding unit. We applied up to two categories from Table 3 to a coding unit. The entire experts' feedback to all other steps were considered as context unit. Using deductive categories from the literature, three different teams of two authors each independently coded a different sequence (*Bob*, *Alice*, *Parker*), as shown in Table 2.

Table 2: Allocation of (A)uthors to coding sequences

Author	CodingBat		FITech		JetBrains	
	Bob	Alice	Eve	Dave	Parker	Jasmine
A1	x	x		x		x
A2	x			x	x	x
A3			x		x	
A4		x	x			

After the initial round of coding, differences were discussed between all authors. We reached a consensus after slightly adapting the category definitions and extending them with anchor examples. These changes reflected what we found in the data, hence, inductive

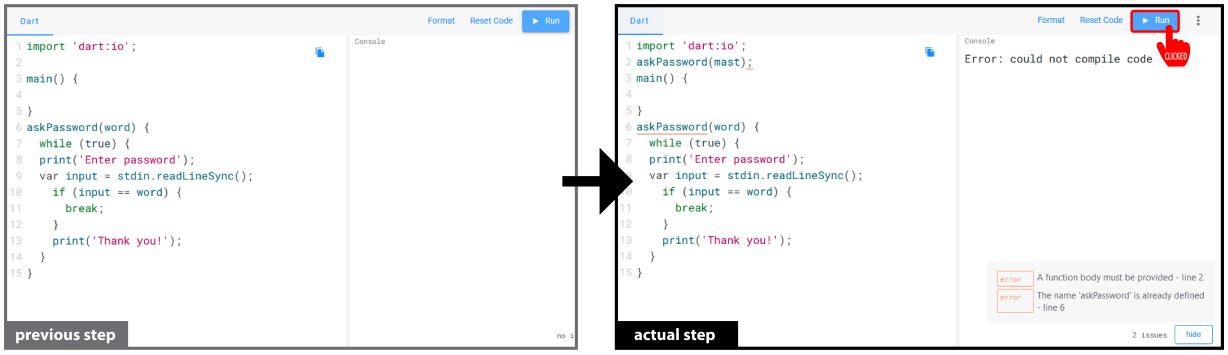

Figure 1: Two consecutive steps from Dave; in the actual step (right), he clicked the RUN-button to receive feedback.

categories were added (e.g., "other categories" in Table 3). Next, all four authors (re-)coded the steps where disagreement occurred in the first round in the *Parker* sequence, and discussed the results until reaching agreement. Finally, two pairs of authors used the slightly adapted coding scheme to recode *Eve* and *Dave*. We reached agreement in a final discussion and adapted the coding scheme as shown in Table 3 to include, e.g., debugging as part of KTC-TPR. Moreover, we decided to code every coding unit twice with respect to its content and format dimension. As a last step, two of the authors coded *Jasmine*.

4 Results: How experts provide feedback to novice programmers

Applying an existing feedback taxonomy to expert feedback led to the deductive-inductive categories that describe how experts give feedback and hints to students, as shown in Table 3. For each category, we provide a detailed definition, along with anchor examples [18]. The results show that most of the experts' feedback can be categorized using Keuning et al.'s feedback taxonomy. However, we also identified new categories based on the data. Next, we elaborate on all of the applied categories and present additional findings regarding the format of the feedback, and expert (dis-)agreement.

Knowledge on Task Constraints. For KTC feedback, we found new subcategories that we subsumed under KTC-TPR. Some of the experts' interventions focused on basic debugging skills, e.g., learners were advised that it is better to comment out (faulty) code instead of deleting it. In addition, we often found instructions to calculate certain parts of the program "by hand", e.g., on paper. Both represent basic procedural programming knowledge not explicitly related to a specific task, for which we introduced new categories in KTC-TPR. We also found some cases where experts recapitulated the task requirements addressed in the step or just repeated what the student did so far ("I see you've successfully created a loop over all the characters in the string, and you're [comparing] each character with the previous largest value." We decided to code these as KTC-TR as we only saw feedback repeating (fulfilled) task requirements.

Knowledge on Concepts. We found only one statement from an expert that could be classified as KC-EXA. For KC-EXP, we only mapped feedback that included concepts related to programming. For instance, explanations to calculate the Fibonacci sequence were categorized as KTC-TR rather than KC-EXP, since this concept was integral to the task and is not related to programming.

Knowledge on Mistakes. Some experts directly addressed syntax errors, while others referred to the error messages given by the compiler (KM-CE). We also labelled feedback on type errors and indentation errors using KM-CE, since the IDE marked them as errors. Some experts gave feedback as soon as the error occurred, whereas other experts waited a few steps before drawing attention to the error. Feedback concerning test results from the IDE was labelled as KM-TF. We extended this category to include cases where experts referred to passed tests, not just failures.

Knowledge on How to proceed. When experts provide hints on how to proceed to fix a specific (syntactic or semantic) error, we labelled them as KH-EC. Feedback that guided learners on the next steps towards the correct solution was categorized as KH-TPS. Hints on how to improve the style or performance of code (KH-IM) were generally given at the end of the sequences, after the learner's solution was already correct. In a few instances, however, KH-IM feedback was given at the very start of the session.

Other Categories. Even though most cases were categorized using the existing feedback categories, the experts' responses also included other feedback elements. For example, some experts acknowledged the learners' progress in the right direction, when a mistake was fixed or when the task was solved correctly. These cases were subsumed under the category MOT, which focuses on the motivation of the learners. For this category, we only labeled the feedback in which the expert explicitly mentioned that they want to motivate the learner ("Motivate the student (on the right track with +)", or if there was a statement addressed directly to the learner with a clear focus on motivation ("Neat! Well done!").

In some cases, experts were uncertain about what the learner was doing, so they explicitly asked for further information (e.g., "ask them to tell me what they are thinking"). We coded these general questions as GETI, and thus as distinct from pedagogically guided, or reflective questions aiming at supporting students. Interestingly, we often observed the GETI category in combination with a "Maybe" decision for giving feedback or not.

Table 3: Category system – *How* experts intervene and provide feedback.

Categ.	Definition	Anchor Examples (with indication of format ⓘ or Ⓡ)
KTC	**Feedback on task constraints**	
TR	Focuses on specific *task requirements* needed for a correct solution. Includes e.g., required paradigms (use of recursion), specific naming conventions for variables, imposed restrictions (no Java API).	Ⓡ"Let's read the exercise again to make sure we understood the assignment correctly." ⓘ"I will try to explain the formula of the fib sequence."
TPR	Hints on *task-processing-rules* provide general information on how to approach the task. This also includes general explanations of how to solve certain types of tasks (e.g. how to proceed with recursive programs). Also includes hints for procedural knowledge in programming such as debugging strategies, or tracing code.	Ⓡ"I would ask him what value would n have to have to reach the recursive call." ⓘ"When you see a red line, it means there is a syntax error." ⓘ"Try running it again. Now what does it do?"
KC	**Feedback on concepts**	
EXP	*Explanations on subject matter*: Explanations or hints for repeating certain programming concepts that are relevant in the context of programming. Other concepts, such as mathematical formulas, which are expected by the task fall into the KTC-TR category	ⓘ"Additional explanation about attributes of objects/strings" ⓘ"Explanation about how isdigit() works and what else could be helpful for the task." ⓘ"The structure of a conditional statement is: if(condition){statement}" Ⓡ"Explain how to do type casts in Python."
EXA	Examples that illustrate programming concepts (e.g. analogies) such as references to course material that contain examples to the concept.	ⓘ"I would remind her of the purpose of semicolons (giving the analogy of semicolons are like periods to sentences)" Ⓡ"check course material to see what if-statements should look like"
KM	**Feedback on mistakes**	
TF	References to *test results* (e.g. provided Unit-Tests).	Ⓡ"Look at the first case that fails here. Why is that?" Ⓡ"Why do you think case 0 and 1 pass but the rest fails?"
CE	Reference to *Syntax Errors* in the code (e.g. Indentation Error, Missing Parenthesis/Semicolon, Type Error) typically identified by a compiler or interpreter	Ⓡ"Try to think about what the error message is trying to tell you." Ⓡ"Think about the values of v3 and v2. Can they be compared? Do they represent the same data value?" ⓘ"Notice how v1 is a str, but v2 is an int. [...] Python is upset because you are comparing a number (0) to a character (a digit)."
SE	Reference to *Semantic Errors* in the solution (e.g. logic error, incorrect operator use)	Ⓡ"What are you trying to achieve with v3.isdigit? Why do you need something like it?" Ⓡ"Do you (really) need to parse your variable into an int?"
SI	Reference to *Style Issues* e.g. if the code is unreadable, certain conventions are not adhered to when naming variables or if the code is badly formatted.	ⓘ"It is better to always introduce a block { } for the then-part." Ⓡ"They [...] should now look at it as a whole and consider if there are any design or style improvements they want to make." Ⓡ"I would ask Parker what v2 and v3 stand for and whether they are good variable names."
PI	Reference to *Performance Issues* e.g. the program has a poor runtime or requires an unnecessarily large amount of memory.	Ⓡ"I might ask if she can think of a better/faster solution, and take it to the next level." Ⓡ"What [is] the running time of this program in relation to n? Would there be any possibility of solving the exercise with a faster program?"
KH	**Feedback on how to proceed**	
EC	Bug-related hints for error correction: Focuses on how to proceed to fixing a specific syntax/semantic error in the learner's program.	Ⓡ"I would ask them to walk me through their process on how they would resolve them." ⓘ"Everything should be lined up at the same column instead of bodies of if/else/while/etc." ⓘ"I would indicate that line 6 needs to be v3 = int(v3)."
TPS	Feedback on Task Processing Steps: Gives hints / encourages reflection on how to proceed to get one step further.	Ⓡ"Complete the code, and ignore these errors for now." ⓘ"That print statement might help you. Perhaps you can use another argument?" ⓘ"Complete the code, and ignore these errors for now."
IM	Focuses on how to proceed to improve the quality of the program (style, performance).	Ⓡ"Can you not combine these two if statements?" ⓘ"You could also choose to have one condition: if (n == 0 || n ==1) { return n }" ⓘ"Variable names often help understand the type of an variable: input_string, one_char, highest_char"
	Other categories	
KMC	Feedback on Meta-cognition: which strategy to use to solve a problem. Also contains offering help, when struggling; not related to specific errors	ⓘ"Think before you type." Ⓡ"I would ask if they had questions about how to proceed."
GETI	The expert does not fully follow what the student is doing and explicitly asks what they think/do/know to get information how to give feedback.	Ⓡ"It is not clear what 'My' refers to here. Do you mean a variable 'My' or do you want to print the word 'My' to the screen?" Ⓡ"No idea, first I want to find out what are her thoughts about this program"
MOT	Feedback that aims to motivate the learner.	ⓘ"Motivate the student (on the right track with +)" ⓘ"I would congratulate them on having passed all the tests and note they have a nice solution."

4.1 Format of expert feedback: Providing information or asking questions

Another crucial finding of our analysis was that human expert feedback – compared to automated programming feedback in learning environments – differed not only regarding its content (as depicted by the categories in Table 3, but also regarding the *format*. For example, some experts explicitly provided information, e.g., "there is a missing semicolon at the end of line 5". Other experts asked guiding questions to encourage learners to think critically about their code or certain concepts (e.g., "What is a good variable name?").

To capture these varying forms and expressions, we extended our categories with an additional dimension: ⓘ for information and Ⓡ for reflective elements. We added the respective dimension to all anchor examples in Table 3. In some long responses, the experts provided feedback that included both of these forms. For example, in the same feedback message, information about an error was provided, followed by a reflective question.

4.2 Experts' agreement

We noted that the experts varied in their feedback approaches. For example, for the same step of a learner's problem-solving process in the Fibonacci task, some experts gave feedback to correct a specific error (KH-EC), while other experts chose to explain the concept of recursion in greater detail (KC-EXP), or suggested troubleshooting using test results as shown by the environment (KM-TF).

However, we also found some steps where the experts largely agreed on how to give feedback, and the feedback only differed with respect to its format. This was the case in the Fibonacci task, when Bob had successfully implemented the two base cases. Once he was done, all experts who decided to (maybe) give feedback provided a hint on collapsing two if-statements into one (KM-IM).

5 Limitations

Even though we instructed the experts to directly express their feedback to the student, some of them rephrased it and directed it to us as researchers. In some cases, this led to challenges in the coding process, as it was not clear which feedback the expert was going to provide to the student. In other cases, the expert feedback was too brief, too abstract, or hypothetically citing additional conditions (e.g., "if the student would do X, or ask for Y, I'd tell them Z"). To prevent some of these challenges, it may be advisable to provide a video of the student solving the problem. Another limitation is that experts may have provided socially desirable, exaggerated or partial responses, meaning responses are not necessarily identical to the feedback they would have provided in an authentic classroom setting. Moreover, the observer's paradox [25] should be taken into account. We also did not assess the experts' qualifications or verify their experience, thus their expertise may be limited after all.

6 Discussion

Mapping the experts' feedback to existing feedback taxonomies [10, 20] is challenging in some cases. The categories from Keuning et al., for example, have been constructed based on the analysis of learning environments. These typically provide automatically generated information on mistakes (KM) and how to correct them (KH) [9, 10]. Our respondents, however, often used guiding questions and reflective elements. For this reason, we added a dimension describing the format of the feedback. It should be noted though that the term *format* has been subject to extensive discussions, as it proved challenging to infer the experts' actual intentions or personal style of giving feedback. In the present form, the data only allowed us to categorize this as feedback format. In the future, this category may be reused and expanded by explicitly asking educators about their intention and strategy.

We also observed that experts' responses often comprised multiple aspects. Hence, we applied multiple categories to capture all of them, e.g., motivational feedback and improvement hints. Yet, it was challenging to identify which aspect experts prioritized and what their underlying strategy was. For example, the feedback "Start with the error message" may be coded as KM-CE-ⓘ as it refers to a syntax error in the code signalled by the learning environment. At the same time, very little information is provided about the specific error, the learner has to read and understand the error message and reflect the existing code so that KM-CE-Ⓡ may also be applicable. Another underlying strategy of the expert may be to guide a learner so that they generally consider the error messages first, before moving on and adding more code (KTC-TPR-ⓘ). In a few cases, the context proved to be helpful, i.e., by looking at feedback on previous steps or on the reason WHY the expert gave feedback.

The results indicate that – unlike automated feedback in learning environments – expert feedback often includes guidance on how to proceed (also known as *feed forward* [8]). While learning environments mostly provide feedback related to syntax or quality issues in the code [9], the aim of expert feedback is supposedly to foster learning and competencies. Moreover, the construction of *feed forward* feedback (e.g., KH) requires significantly more effort than implementing unit tests. When it comes to experts giving individual feedback, it is necessary to analyze the process and formulate different feedback messages for each erroneous or suboptimal strategy. Fortunately, generative AI tools offer more and more potential for future applications and individual feedback [1, 11, 15, 22].

7 Conclusions and Future Work

The study presented in this paper is an initial step towards providing insights into *how* experts provide feedback on steps learners take when solving introductory programming tasks. We investigated the extent to which expert feedback can be categorized using existing feedback taxonomies [10, 20] by analyzing 210 feedback messages gathered from a survey with 47 experts (experienced programming educators from different countries and educational sectors). As part of the survey, we presented authentic programming sequences of learners working on an introductory programming task. We asked the experts *how* to give feedback on each step.

We qualitatively analyzed the experts' feedback using a deductive-inductive approach resulting in a coding scheme with 15 categories for *how* experts provide feedback. The results show a degree of variation among educators, meaning we did not identify specific, or coherent feedback strategies. In general, the existing feedback taxonomies [10, 20] could be applied to the expert feedback. We extended them to include two more categories (GETI and MOT) to cover experts' asking for more information from the learners, and motivational feedback. In addition, we identified varying formats: expert feedback was either presented as information (ⓘ), or as reflective element (Ⓡ, e.g., guiding question).

However, it was sometimes challenging to decide what the expert's strategy was. Even though we considered all of the experts' responses to a sequence as context unit, we could not fully identify their intentions, or pedagogical reasoning. To gain deeper insights into expert feedback, future research should explore how the state of the program code influences the type of feedback provided by experts. For example, do experts tend to offer specific types of feedback when the student's code is in a particular state or shows specific characteristics or errors? Previous research has already explored the reasons *why* experts decide to give feedback [16]. In future work, we want to investigate if there are correlations between experts' reasons why to give feedback, their feedback type and format. Future work may use qualitative interviews to achieve this. Overall, we consider it critical to understand expert feedback before it can be modeled or automated as a part of learning environments.

References

[1] Imen Azaiz, Natalie Kiesler, and Sven Strickroth. 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) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 31–37. doi:10.1145/3649217.3653594

[2] Brett A Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, et al. 2019. Compiler error messages considered unhelpful: The landscape of text-based programming error message research. *Proceedings of the 2019 Working Group Reports on Innovation and Technology in Computer Science Education* (2019), 177–210. doi:10.1145/3344429.3372508

[3] Tyne Crow, Andrew Luxton-Reilly, and Burkhard Wuensche. 2018. Intelligent Tutoring Systems for Programming Education: A Systematic Review. In *Proceedings of the 20th Australasian Computing Education Conference* (Brisbane, Queensland, Australia) *(ACE '18)*. ACM, New York, 53–62. doi:10.1145/3160489.3160492

[4] Veronica Cucuiat and Jane Waite. 2024. Feedback Literacy: Holistic Analysis of Secondary Educators' Views of LLM Explanations of Program Error Messages. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 192–198. doi:10.1145/3649217.3653595

[5] Sidney K. D'Mello, Blair Lehman, and Natalie Person. 2010. Expert Tutors Feedback Is Immediate, Direct, and Discriminating. *Proceedings of the 23rd International Florida Artificial Intelligence Research Society Conference, FLAIRS-23* (May 2010). https://cdn.aaai.org/ocs/1215/1215-7820-1-PB.pdf

[6] Yihuan Dong, Preya Shabrina, Samiha Marwan, and Tiffany Barnes. 2021. You Really Need Help: Exploring Expert Reasons for Intervention During Block-Based Programming Assignments. In *Proceedings of the ACM Conference on International Computing Education Research (ICER)* (Virtual Event, USA) *(ICER 2021)*. ACM, New York, 334–346. doi:10.1145/3446871.3469764

[7] John Hattie. 2009. *Visible Learning: A Synthesis of over 800 Meta-Analyses Relating to Achievement*. Routledge, London ; New York. https://doi.org/10.4324/9780203887332

[8] John Hattie and Helen Timperley. 2007. The Power of Feedback. *Review of Educational Research* 77, 1 (2007), 81–112. doi:10.3102/003465430298487

[9] Johan Jeuring, Hieke Keuning, Samiha Marwan, Dennis Bouvier, Cruz Izu, Natalie Kiesler, Teemu Lehtinen, Dominic Lohr, Andrew Peterson, and Sami Sarsa. 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) *(ITiCSE-WGR '22)*. ACM, New York, 95–115. doi:10.1145/3571785.3574124

[10] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Trans. Comput. Educ.* 19, 1, Article 3 (sep 2018), 43 pages. doi:10.1145/3231711

[11] Natalie Kiesler, Dominic Lohr, and Hieke Keuning. 2023. Exploring the Potential of Large Language Models to Generate Formative Programming Feedback. In *2023 IEEE Frontiers in Education Conference (FIE)*. IEEE, College Station, TX, USA, 1–5. doi:10.1109/FIE58773.2023.10343457

[12] Avraham Kluger and Angelo DeNisi. 1996. The Effects of Feedback Interventions on Performance: A Historical Review, a Meta-Analysis, and a Preliminary Feedback Intervention Theory. *Psychological Bulletin* 119 (03 1996), 254–284. doi:10.1037/0033-2909.119.2.254

[13] Raymond W Kulhavy and William A Stock. 1989. Feedback in written instruction: The place of response certitude. *Educational psychology review* 1 (1989), 279–308. doi:10.1007/BF01320096

[14] Dominic Lohr and Marc Berges. 2021. Towards Criteria for Valuable Automatic Feedback in Large Programming Classes. In *Hochschuldidaktik Informatik HDI 2021*. Jörg Desel, Simone Opel, Dortmund, 181–186. https://delfi-tagung.de/fileadmin/FG/BBI/user_upload/5410_HDI-Tagungsband_web.pdf

[15] Dominic Lohr, Hieke Keuning, and Natalie Kiesler. 2025. You're (Not) My Type-Can LLMs Generate Feedback of Specific Types for Introductory Programming Tasks? *Journal of Computer Assisted Learning* 41, 1 (2025). doi:10.1111/jcal.13107

[16] Dominic Lohr, Natalie Kiesler, Hieke Keuning, and Johan Jeuring. 2024. "Let Them Try to Figure It Out First" - Reasons Why Experts (Do Not) Provide Feedback to Novice Programmers. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 38–44. doi:10.1145/3649217.3653530

[17] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018 Companion)*. Association for Computing Machinery, New York, NY, USA, 55–106. doi:10.1145/3293881.3295779

[18] Philipp Mayring. 2001. Combination and integration of qualitative and quantitative analysis. In *Forum Qualitative Sozialforschung/Forum: Qualitative Social Research*, Vol. 2. Art. 6. https://doi.org/10.17169/fqs-2.1.967

[19] Philipp Mayring. 2014. *Qualitative Content Analysis: Theoretical Foundation, Basic Procedures and Software Solution*. Technical Report. Leibniz Institute for Psychology, Klagenfurt, Austria. https://doi.org/10.1007/978-94-017-9181-6_13

[20] Susanne Narciss. 2008. Feedback strategies for interactive learning tasks. *Handbook of research on educational communications and technology* 3 (2008), 125–144.

[21] Claudia Ott, Anthony Robins, and Kerry Shephard. 2016. Translating Principles of Effective Feedback for Students into the CS1 Context. *ACM Transactions on Computing Education* 16, 1 (2016), 1–27. https://dl.acm.org/doi/10.1145/2737596

[22] James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Petersen, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. 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) *(ITiCSE-WGR '23)*. Association for Computing Machinery, New York, NY, USA, 108–159. doi:10.1145/3623762.3633499

[23] Yizhou Qian and James Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.* 18, 1, Article 1 (oct 2017), 24 pages. doi:10.1145/3077618

[24] Hemilis Joyse Barbosa Rocha, Evandro De Barros Costa, and Patricia Cabral De Azevedo Restelli Tedesco. 2023. Helping to Provide Adaptive Feedback to Novice Programmers: A Framework to Assist the Teachers. In *2023 18th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, Aveiro, Portugal, 1–6. doi:10.23919/CISTI58278.2023.10212000

[25] Fritz Jules Roethlisberger and William J. Dickson. 1939. *Management and the Worker*. Harvard University Press, Cambridge.