"Let Them Try to Figure It Out First" - Reasons Why Experts (Do Not) Provide Feedback to Novice Programmers

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

A recent ITiCSE working group investigated when and how experts give feedback and hints at steps novice programmers take when solving programming problems. Based on the feedback literature and an analysis of expert feedback on steps, the working group designed guidelines for when and how to give feedback. The feedback provided by educators using these guidelines on a number of sequences of student steps varied a lot. In this paper, we try to answer the question of *why* educators give feedback at particular steps to novice learners of programming. We prepared six authentic sequences of student steps when solving an introductory programming task. The preprocessed sequences were used in a survey to gather information about when and why an expert would give feedback. Respondents annotated each step from one sequence with if and why they would give feedback at that step. Our survey received 47 responses. We qualitatively analyzed the responses, resulting in a coding scheme consisting of 19 different reasons for why experts intervene (or not) when novice learners work on introductory programming tasks. We found a considerable variety of reasons experts give for when and how to help students with feedback and hints. Also, sometimes one expert uses a reason at a step to explain why they do intervene, and another expert uses the same reason at the step to not intervene. The categories of experts' feedback indicators will pave the way for several future studies and applications, including learning systems trying to resemble expert feedback strategies.

CCS CONCEPTS

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

KEYWORDS

Learning programming, expert feedback, feedback rationale, feedback guidelines, novice programmers

ITiCSE 2024, July 8–10, 2024, Milan, Italy

ACM ISBN 979-8-4007-0600-4/24/07.
https://doi.org/10.1145/3649217.3653530

ACM Reference Format:
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 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024), July 8–10, 2024, Milan, Italy.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3649217.3653530

1 INTRODUCTION

Challenges in introductory programming for both students and educators are manifold. Educators may struggle with large classes, limited resources, or increasingly heterogeneous groups of students [28]. At the same time, students are facing cognitively complex tasks [10, 14], and unrealistic expectations [21].

Programming courses almost always offer tasks in which students need to develop a program in some programming language to solve a problem. During these learning activities, students may require more information about the task requirements, previous actions, mistakes, or how to proceed. Such feedback is crucial for student learning [7, 32]. There is literature on general guidelines for feedback, but how does this translate to programming education contexts, and solving programming problems step by step?

A recent working group [8] at the *Innovation and Technology in Computer Science Education* (ITiCSE) conference tackled this question by focusing on *when* students need feedback and hints in the programming process, and *how* feedback should be given. Although the report provides recommendations on this from an expert perspective, they conclude more experts are needed to reach consensus regarding which feedback to provide to learners. Moreover, it remained unclear why experts provide specific feedback [8].

This paper addresses this question by reusing, transforming, and expanding the datasets from the working group to gather information from experts on reasons to give feedback to novice programmers. The **goal** is to understand *why* educators provide feedback in introductory programming contexts. Our **contributions** are a methodology to process extensive keystroke data from students solving programming tasks in learning environments, 6 comprehensive student sequences, and a set of inductive categories reflecting experts' reasons to provide feedback to students in this context. The results can be used in future research on expert feedback, its quality, suitability to learners' informational needs, and how it compares with feedback given by peers, learning environments, or AI models.

2 RESEARCH ON FEEDBACK

Feedback is considered one of the most influential factors in learning success [6, 7, 16, 25]. According to Hattie and Timperley's model [7], effective feedback answers the questions of "where am I going", "how am I going", and "where to next", which operate at four levels: (1) *task*, (2) *process*, (3) *self-regulation* and (4) *self*. Whereas (1) deals with how well the student understands the task itself, (2) focuses on the process to solve it, (3) is about the student monitoring and regulating their actions, and (4) is about personal evaluation.

Feedback can be defined as a multidimensional construct in which determinants like the feedback source, type, modality, or timing can influence whether feedback is perceived as valuable or not [25]. The source of feedback can be humans (teachers, TA's, peers) and systems (automated feedback in learning environments). The latter's goal is usually to simulate a human tutor [33]. Research exploring the correlation between the timing of feedback and its impact on learning and performance has yielded inconsistent results. Shute [32] reports that many field studies show the positive effect of *immediate* feedback, while lab studies show this effect for *delayed* feedback. Narciss [25] provides an extensive classification of the content of feedback in the context of interactive instruction.

Feedback has also been studied in the context of programming education (e.g. [2, 3, 5, 8, 9, 13, 18, 20, 26, 29, 31]). Ott et al. [26] explored the applicability of Hattie and Timperley's feedback model [7] in the CS1 context and determined that automated feedback contributes to the first three levels. The authors excluded the *self level* (4), as they deemed it lacking a significant impact on enhancing learning performance. The researchers devised a roadmap outlining effective feedback practices for various levels and stages based on Hattie and Timperleys' feedback model. Keuning et al. [9] developed a classification of automated feedback for the domain of programming exercises, extending Narciss' feedback content classification [25]. The authors found that the majority of the feedback that tools give is based on pointing out mistakes. Hao et al. [5] conducted a controlled quasi-experiment to study the effects of different types of programming feedback (correct/incorrect, or more elaborated feedback based on test cases). They found that students receiving more detailed feedback performed significantly better than those only receiving binary feedback.

A recent ITiCSE working group [8] (WG) 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. Dong et al. [3] investigated tutor's interventions during introductory programming assignments in a block-based environment. They found some 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*.

To conclude, there is an extensive body on feedback research, also in the context of (introductory) programming education. Some of these studies go beyond providing feedback and look more closely at its content and timing. However, we found very few studies that looked at reasons why feedback is or is not given at a particular point in the programming process.

3 METHODOLOGY

Novice learners of programming often benefit from timely feedback and hints. But what are the reasons why experts give feedback and hints? This paper investigates the following research question:

RQ *Why do experts provide feedback (or not) to novice programmers who are solving programming tasks?*

3.1 Preprocessing of Student Data

3.1.1 Selection of Student Data. To gather the experts' reasons on why to give feedback, we used available (published) datasets [15] containing students' steps while solving introductory programming tasks. The datasets were collected through online learning environments, and published by the 2022 ITiCSE WG [8].

The FITech dataset [19] was obtained from an online programming course at Aalto University. FITech contains keystroke-level code edit actions as well as *run*, *submit*, and *request help* actions captured from an online IDE embedded in a course platform for distance learning. The data is from a course that covers the principles of programming using the Dart language and contains 64 different small programming tasks. We use the data for one of the two simple tasks: reading input until it matches a required value.

CodingBat is a learning environment for programming, developed by Parlante [27]. The CodingBat dataset contains data from novice programmers working on Java tasks gathered via thinking-aloud experiments in a controlled environment [11, 12]. The dataset comprises two tasks (Fibonacci, factorial of n), and 16 student sequences at token-level granularity.

We used an additional dataset not collected by the WG: the TaskTracker dataset from Lyulina et al. [22]. It consists of 148 students solving 6 different programming tasks in 4 different programming languages, for which fine-grained data is collected through a plugin for IntelliJ-based IDEs. We used the Python programs from this dataset, another commonly taught programming language. We have limited information on how the dataset was collected.

3.1.2 Selection of Tasks and Sequences. All selected datasets contain authentic student sequences. We used the following guidelines to select particular sequences:

- The task is designed for novice learners of programming.
- The length of a 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.

3.1.3 Selection of Steps. The granularity of the selected sequences was too fine-grained to show them to teachers. For example, even though the FITech dataset had been preprocessed by the WG, every student solution still comprised 144.1 steps on average with a standard deviation of 120.7 steps. In particular sequences containing a number of errors, from students regularly asking for feedback from the learning environment, are long. For practical purposes, we wanted sequences with at most 25 steps. We reduced the number of

Table 1: Overview of data sequences.

Name	Source	Id	Task	Steps	
Bob	CodingBat	B02	Fibonacci	9	(full)
Alice	CodingBat	B05	Fibonacci	10	(full)
Eve	FITech	148	Password	23	(partial)
Dave	FITech	538	Password	15	(full)
Parker	TaskTracker	tt1	MaxDigit	12	(full)
Jasmine	TaskTracker	tt2	MaxDigit	22	(full)

steps while preserving key steps and their meaning in each selected sequences. We *kept steps* in the data sequence for the following reasons:

- When the initial line(s) of code were provided by the learning environment.
- When a student executes code, and/or requests feedback (e.g., via a compile, run, or hint-button).
- When a student pastes in code.
- When a student pauses for 2 minutes or longer before writing the next step, both the step before the pause and the step after the pause are kept.
- When a student continuously writes code and finishes a line of thought, we keep a snapshot of every finished line of code including open and closing parentheses that might appear on the previous or next line. When a student pastes in multiple lines of code at once instead of typing them, multiple lines are added as one step.
- When a student makes an error (e.g., a typo) within a keyword, we keep the step containing the first instance of the error to understand its origin and nature; if the error is due to formatting, we keep the step before and after the formatting

Steps were removed from the sequence for the following reasons:

- When a student continuously writes/inserts code without a pause or deletion (even if errors are made), and thus continues a line of thought, we remove intermediate steps until a line is finished (even if there are minor errors that are immediately corrected).
- When the same step occurs multiple times with the same timestamp in a dataset, we remove the repeated step.
- When a student executes a program twice in a row, we remove the second step.

Applying these criteria to the selected sequences gave sequences in which the majority of the steps add or change (part of) a line in a program. See Fig. 2 for an example of a step where a student completed a line and pressed the Go button.

The preprocessing of data resulted in the selection of six sequences (which can be either the full or the partial sequence) for three different tasks in three different programming languages, as shown in Table 1 and 2[1].

3.2 Survey Design

The survey first asked the participants a number of general questions about job title, years of experience in teaching programming courses, school level, the main country of teaching, and in which programming languages they feel comfortable giving feedback (C# and Java, or Python, or all of these). Since at the level of our datasets Dart is almost indistinguishable from C# and Java, and 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.

[1]The sequences are available at https://github.com/Programming-Steps-Working-Group-2022/Expert-Reasons-ITiCSE-24.

Table 2: Task details.

Exercise	Lang.	Description	Concepts
Fibonacci	Java	Compute the n-th Fibonacci number using recursion.	methods, conditionals, recursion
Password	Dart	A function asking the user for a password until they enter the correct one (given as parameter).	methods, parameters, loops, conditionals, input/output
MaxDigit	Python	Given a string containing only digits, find and print the largest digit.	loops, conditionals, input/output

After the general questions, we would randomly offer 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 describe a fictional student solving the task, see for an example Fig. 1. We developed six such personas based on our knowledge of the contexts in which the datasets from which we selected the sequences were collected.

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.

Figure 1: A description of a persona.

We gave the following instructions for answering the remaining questions:

> *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. 2 gives an example of a step that was shown to the participants. We always present the preceding step to indicate the students' progress. For each step, we ask 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).

3.3 Data Gathering from Experts

Before distributing our survey, our institutional ethics committee approved our research proposal. All participants gave consent for using 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 several 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 the survey to various (international) colleagues of the authors who were considered experts (i.e., people with years of teaching experience in programming classes).

3.3.2 Description of the Sample. We received 47 responses; most came 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 some spread: the population standard deviation was 11.65.

The feedback of the respondents was distributed over the various sequences (within parentheses the number of answers to the *why* question for each of these sequences): Bob 6 (38), Alice 9 (76), Eve 10 (177), Dave 5 (37), Parker 13 (102), and Jasmine 4 (34). In total, we analyzed 464 reasons why or why not to give feedback. 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 the experts' responses, we applied a qualitative analysis technique [24] to the open-ended questions in which experts indicated *why* they provided feedback or not. In this paper, we only analyze the reasons for experts to give feedback. We do not evaluate the decisions (Yes/Maybe/No) they made based on this reason, which is future work.

The starting point of this iterative process was an exploration of the material while trying to apply the deductive categories developed as part of prior work [8]. Even though several categories seemed applicable, a first run through the responses quickly confirmed the need to develop new, inductive categories summarizing and abstracting the experts' reasons for giving feedback [23, 24]. These were constructed for the first sequence (Alice). At the same time, not all categories established in the working group report could be reused, as experts did not refer to all of them.

Three of the authors coded the experts' replies independently starting with a small subset of the material (Alice, 76 reasons). Then the applied codes were compared and discussed until reaching consensus to ensure reliability. Two of the authors were coders in the entire coding process. An extended version of the category system was then applied to the next sequence (Bob, 38 reasons), before refining the category definitions and anchor examples in alignment with the experts' explanations. This process was continued when moving on to the next task and the respective two sequences (Dave, 37 reasons and Eve, 177), thereby specifying the category definitions further, and adding new categories for new reasons. After reconciling half of the codings for Eve (step 1-11), the coders restructured some of the categories and recoded the sequences. When comparing the revised codings for Eve (step 12-22), the agreement had improved significantly. Then the last task was coded (Parker, 102 and Jasmine, 34), starting with responses to Parker's sequence. It resulted in the addition of one more category (TYPE). For the last sequence (Jasmine), no new categories were added.

For Parker's sequence with 102 responses (21.98% of the data), we calculated the intercoder-reliability by using Cohen's κ [1]. With 0.68, the agreement was substantial [17].

4 RESULTS: WHY EXPERTS GIVE FEEDBACK

Table 3 presents the deductive-inductive categories describing reasons why or why not experts decide to give feedback and hints to students. For each category, we provide a detailed definition, along with anchor examples.

The first five categories (presented in italics) reflect deductive categories that were identified in the literature [8], and recognized in the expert responses. The 14 new, inductively-built categories were constructed based on the material alone. They comprise a great variety of reasons why experts consider intervening or not. Several categories (e.g., SYNE, TYPEE, ERROR) relate to actual errors in the student code. In contrast, other categories refer to minor aspects that could be improved but do not necessarily cause an error (e.g., STYLE, PERF). Additional indicators for experts to (not) intervene were related to the occurrence of the student step within the sequence. Intervening when a student had just started the problem-solving process (START), for example, was very unlikely

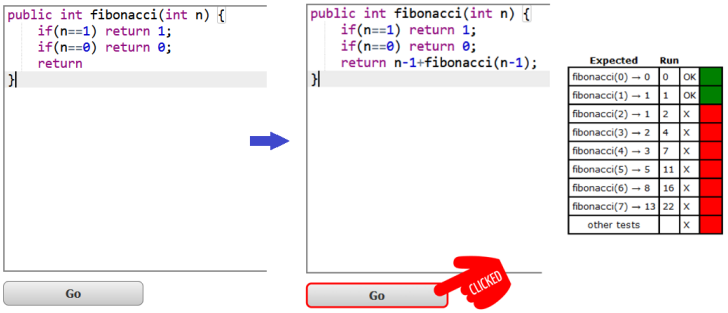

Figure 2: Two consecutive steps from Alice; in the second step she clicked the Go-button to receive feedback.

Table 3: Categories for the reasons *why* experts intervene and provide feedback (or not).

Category	Definition	Anchor Examples
Deviation from specification (DFS)	Inconsistencies with the task description and requirements, e.g., changing required function signature.	"Bob's code fails to be consistent with the definition of the Fibonacci function." "The two lines are both irrelevant to the specified task. " "Eve is limiting the type of password to numbers, which should not be done."
Trial and Error behaviour (T&E)	Once it becomes clear the edits are guessing – not experimentation.	"typing without thinking" "looks like the student is using trial-and-error to find the bug" "random programming"
Feedback request (REQUEST)	When a student (would) ask for help.	"If the student asked for help." "Only when being asked."
Task completion (COMPLETE)	When a student completes all steps of a task and achieves a correct solution.	"Celebrate success!" "The solution is correct."
Logical error (LOGE)	Program errors that causes the program to operate incorrectly.	"The logic is wrong." "It seems she wants to sort the digits, but she only sorts a list of length 1"
Syntax error (SYNE)	Errors related to spelling, punctuation, parenthesis, indentation, etc. causing a compiler error.	"missing semicolon" "The last 'return' statement fails to return a value, so the compiler should report that error."
Type error (TYPEE)	Operations on values that are not of the appropriate data type, e.g., adding a string to an integer.	"they will eventually run into TypeError: '>' not supported between instances of 'str' and 'int'" "trying to verify the string is a digit before doing the comparison, which will fail regardless because of the previous type error."
Unspecified or multiple Errors (ERROR)	When there is no specific information about the error the expert is referring to, or if there are multiple errors.	"the error message now shows another type of error, but in the same line" "the new line is wrongly placed and does not add to fixing the other problems." "Too many problems demotivate beginners."
Style issues (STYLE)	Elements in the code that do not cause a compile error, but should be improved, e.g., lack of meaningful nomenclature, or comments.	"The two if statements can be combined into 1." "it is better to always introduce a block for the then-part" "I would reorder or merge the two cases"
Performance issues (PERF)	When the performance or run-time of an algorithm could be improved.	"I might ask if she can think of a better/faster solution, and take it to the next level (an iterative approach instead of a recursive one)." "This is a good point to talk about running time of the program [...]."
Start of problem-solving (START)	When a student just started the problem-solving process, thus the first step is displayed.	"I think it's too early for feedback. " "They have not written enough code."
Long pause (PAUSE)	When a student does nothing for more than two minutes.	"after 4 minutes without progress, a hint is needed about how to continue."
Progress (PROGR)	Encourage students' progress despite minor things that could be improved; not disrupt the thinking process; wait to see if the student recognizes their mistakes by themselves; or if there is good progress, or significant improvements.	"Asking her to do X would not encourage her progress" "I would wait and see how she continues." "let them go on" "I want her to find out the problem herself" "Elimination of errors, good progress."
Deteriorate solution (STEPBACK)	When student deletes reasonable code, or edits code that had previously been correct.	"The edit did not bring the student any closer to a working implementation" "They completely deleted their reasonable attempt at the return statement instead of revising it."
Existing feedback (EXFEED)	When there is existing feedback, which the student has received or which they will receive, e.g., error messages from the learning environment, the compiler, or the educator.	"The IDE feedback is giving the proper feedback" "the tool already gives feedback" "Previously reported on this milestone and hinted at this issue." "Error message is sufficient."
Uncertainty (UNC)	When the expert is uncertain or undecided about the reasons to provide feedback, or when they do not understand what a student does.	"Unsure yet what they are going to do with v2." "I have no idea what she is trying to do." "I cannot infer what her thought process is from the first line alone."
Confidence in the learner (TRUST)	Assumptions (and their reasoning) about the student's likelihood to succeed by themselves (or not), or the extent to which students can make progress.	"No reason to assume she won't get this" "The student should be able to figure this out for himself" "Does he understand the notion of recursion?" "I'm worried that Parker is confused about what v2 and v3 stand for."
Judging learner's state of mind (STATEOM)	When explicitly making unreasonable assumptions about the students' state of mind, or emotions, cognitive processes, or abilities.	"I think she is panicking.", "the student is officially lost" "I probably need to write this program for her. She is lost" "they get too far off track and become too overwhelmed."
Personal aspects (PERSONA)	Explicitly relating to aspects mentioned in the persona provided for the sequence.	"Bob sounds competent enough and so far so good." "This is a crucial step, since she struggled with recursion [...]."

(in 1 response). If students took pauses longer than two minutes (PAUSE), however, it was a reason to provide feedback or hints.

Students making progress or improvements in their code was one of the most frequent indicators (PROGR, in 153 responses) for experts not to intervene, and exclusively not to intervene. Many experts wanted to let students try to figure out the mistake by themselves first, before interrupting them. This way, students can experience competency. If students deleted reasonable code and deteriorated their solution (STEPBACK), it was a common reason for experts to provide feedback. In some cases, experts expressed their uncertainty (UNC) about the student's progress as an indicator and reason to intervene.

The three remaining categories in Table 3 all refer to personal aspects, somewhat reflecting the "human" dimension of competency-based teaching and learning [4, 30]. This personal note was also introduced by the experts. The category TRUST refers to the expert's confidence in the learner, and whether or not they believe the student can solve the problem or error. A related category (STATEOM) refers to the experts uttering judgments about the students'

state of mind, emotions, cognitive processes, or abilities without explicitly giving a reason for that judgment. The last category related to the human component is PERSONA. It was applied when the experts explicitly referred to aspects mentioned in the persona's description provided as part of the instructions for this study.

In addition, we used three categories not referring to reasons why experts provide feedback to novice programmers (not part of Table 3). These were used to classify other, somewhat unrelated comments (IGNORE), responses expressing confusion about the learning environment or the screenshots in the survey tool (CONF), or when no reason was provided at all (NOR).

5 DISCUSSION

Through a qualitative analysis of 464 experts' responses to six authentic sequences of introductory programming student steps, a category system was derived, as presented in Table 3. Our study extends the 2022 ITiCSE working group results (see section 2) by exploring *why* experts choose to give feedback (or not). Five of the initial categories (DFS, T&E, REQUEST, COMPLETE, LOGE) from the working group [8] were adopted, but some were discarded as experts did not refer to those aspects, or due to the lack of subgoals in the task description. To account for specific error types, we expanded the error categories to address syntax (SYNE), type (TYPEE), logic (LOGE), and multiple errors (ERROR). Due to the substantial interrater-reliability, we assume the coding scheme can be applied to other datasets. At the same time, other reasons to intervene (or not) can likely be added, as this seems to partially depend on the programming language (e.g., type errors).

In some cases, the identification of the expert's reasoning was challenging due to their brevity. For example, when the expert mentioned an "error", it was unclear whether they were referring to errors in the code or the error message from the programming environment (EXFEED).

We also observed patterns among experts. For instance, experts preferred waiting until the task was completed before intervening on performance errors (PERF). This was different for syntax errors or when multiple errors occurred. In these cases, some experts chose to intervene at the first instance of the error(s), while others opted to wait a few steps. However, the experts did not explicitly elaborate on why they intervened early on some errors but not on others. We hypothesize that it is due to the severity of the error. Other patterns include the rationale to let students continue to work on a problem (PROGR, "let's observe the unfolding situation") which was always a reason not to intervene. Another aspect worth mentioning is some experts stating that it was challenging to justify why (not) to intervene at a certain step, as these decisions are often based on intuition and experience.

Relating our work to Hattie and Timperley's model [7] shows that we can loosely align the four focus levels of feedback to our reasons to give or withhold feedback: DFS, LOGE, SYNE, TYPEE, ERROR, STYLE, PERF, and STEPBACK relate to the *task level*, START, T&E, PAUSE, PROGR, and UNC to the *process level*. REQUEST, TRUST, and STATEOM relate to the *self-regulation level*, and PERSONA to the *self level*. However, it should be noted that the categories we developed were all created in the context of a student solving a task. This is different from Ott et al. [26], for example, who map the four feedback levels to the CS1 context, while considering the levels of concept understanding and the course.

The framework can also be used as a basis for further research on feedback in programming education. For instance, it can be used in studies investigating the effectiveness of interventions based on various criteria. Researchers may explore how the categories (or combinations thereof) describing an expert's reasoning map the learner's actual informational needs. Moreover, the student sequences can be reused to investigate other behaviors students exhibit when solving programming tasks in learning environments.

5.1 Limitations

A limitation of this study is the number of investigated student sequences. Finding, accessing, understanding, and preprocessing six sequences representing three different programming languages was a challenge due to the lack of open data resources. During the replay of the students' steps in FITech's learning environment, we also noticed that the version had changed. As a consequence, we could not guarantee that the system's error messages as displayed in the screenshots were identical to those the students received when working on the password task. Another limitation is that the datasets do not comprise students' thinking, gestures, or facial expressions. These indicators could have influenced the feedback of the experts. Moreover, the limitations of self-reporting (e.g., social desirability, exaggerations, omissions) apply to the experts' responses. Finally, the survey design may have caused experts to not remember the task when giving feedback, as they could not go back to the instruction page. However, we did not observe related responses indicating such misunderstandings.

6 CONCLUSIONS AND FUTURE WORK

The presented study is the first of its kind to investigate the reasons why experts provide feedback in introductory text-based programming contexts. We selected and preprocessed six authentic student sequences gathered in three different online learning environments to capture and represent students' progress on beginner programming tasks. In an online survey, their progress was displayed step-by-step, illustrated with screenshots. We recruited 47 experts to indicate for each step whether they would intervene and why. The provided reasons were qualitatively analyzed using a deductive-inductive approach to develop a coding scheme. As a result, we identified 19 different reasons experts use to decide whether or not to intervene in a student's problem-solving process. While these reasons address several types of errors, they also refer to students' actions, uncertainty, or personal aspects of the learner.

The gathered expert responses offer several opportunities for future research: Are there correlations between the reasons why experts provide feedback, and the type of feedback given? How does the experts' background influence their feedback reasoning and strategy? For example, do more experienced programming instructors provide feedback later in the process than educators with less experience? Are there differences regarding the experts' background? In the context of (adaptive) learning environments or systems, it will be interesting to investigate to what extent these indicators can be automated. Therefore, this work is not only valuable for educators but also for people designing learning systems.

REFERENCES

[1] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[2] Barbara Di Eugenio, Davide Fossati, Stellan Ohlsson, and David Cosejo. 2009. Towards explaining effective tutorial dialogues. In *Annual Meeting of the Cognitive Science Society*. Citeseer, 1430–1435.

[3] 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. https://doi.org/10.1145/3446871.3469764

[4] L. Dee Fink. 2013. *Creating Significant Learning Experiences: An Integrated Approach to Designing College Courses* (revised and updated edition ed.). Jossey-Bass, San Francisco.

[5] Qiang Hao, David H Smith IV, Lu Ding, Amy Ko, Camille Ottaway, Jack Wilson, Kai H Arakawa, Alistair Turcan, Timothy Poehlman, and Tyler Greer. 2022. Towards understanding the effective design of automated formative feedback for programming assignments. *Computer Science Education* 32, 1 (2022), 105–127.

[6] John Hattie. 2009. *Visible Learning: A Synthesis of over 800 Meta-Analyses Relating to Achievement*. Routledge, London ; New York.

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

[8] 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. https://doi.org/10.1145/3571785.3574124

[9] 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. https://doi.org/10.1145/3231711

[10] Natalie Kiesler. 2020. 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) *(ITiCSE '20)*. ACM, New York, 459–465. https://doi.org/10.1145/3341525.3387419

[11] Natalie Kiesler. 2022. Dataset: Recursive problem solving in the online learning environment CodingBat by computer science students. Online. https://doi.org/10.21249/DZHW:studentsteps:1.0.0 Datenerhebung: 2017. Version: 1.0.0. Datenpaketzugangsweg: Download-SUF. Hannover: FDZ-DZHW. Datenkuratierung: İkiz-Akıncı, Dilek.

[12] Natalie Kiesler. 2022. *Daten- und Methodenbericht Rekursive Problemlösung in der Online Lernumgebung CodingBat durch Informatik-Studierende*. Technical Report. https://metadata.fdz.dzhw.eu/public/files/data-packages/stu-studentsteps$/attachments/studentsteps_Data_Methods_Report_de.pdf

[13] Natalie Kiesler. 2023. Investigating the Use and Effects of Feedback in CodingBat Exercises: An Exploratory Thinking Aloud Study. In *2023 Future of Educational Innovation-Workshop Series Data in Action*. 1–12. https://doi.org/10.1109/IEEECONF56852.2023.10104622

[14] Natalie Kiesler. 2024. *Modeling Programming Competency : A Qualitative Analysis*. Springer International Publishing, Cham. 165 pages. https://doi.org/10.1007/978-3-031-47148-3

[15] Natalie Kiesler, John Impagliazzo, Katarzyna Biernacka, Amanpreet Kapoor, Zain Kazmi, Sujeeth Goud Ramagoni, Aamod Sane, Keith Tran, Shubbhi Taneja, and Zihan Wu. 2023. Where's the Data? Exploring Datasets in Computing Education. In *Proceedings of the ACM Conference on Global Computing Education Vol 2* (Hyderabad, India) *(CompEd 2023)*. ACM, New York, 209–210. https://doi.org/10.1145/3617650.3624951

[16] 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. https://doi.org/10.1037/0033-2909.119.2.254

[17] J Richard Landis and Gary G Koch. 1977. The measurement of observer agreement for categorical data. *biometrics* (1977), 159–174.

[18] Nguyen-Thinh Le. 2016. A classification of adaptive feedback in educational systems for programming. *Systems* 4, 2 (2016), 22.

[19] Teemu Lehtinen. 2022. FITech dataset. https://github.com/Programming-Steps-Working-Group-2022/public-datasets/tree/main/FITech

[20] 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.

[21] Andrew Luxton-Reilly. 2016. Learning to Program is Easy. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (Arequipa, Peru) *(ITiCSE '16)*. ACM, New York, 284–289. https://doi.org/10.1145/2899415.2899432

[22] Elena Lyulina, Anastasiia Birillo, Vladimir Kovalenko, and Timofey Bryksin. 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) *(SIGCSE '21)*. ACM, New York, 495–501. https://doi.org/10.1145/3408877.3432534

[23] Philipp Mayring. 2001. Combination and integration of qualitative and quantitative analysis. In *Forum Qualitative Sozialforschung/Forum: Qualitative Social Research*, Vol. 2. Art. 6.

[24] Philipp Mayring. 2014. *Qualitative Content Analysis: Theoretical Foundation, Basic Procedures and Software Solution*. Technical Report. Leibniz Institute for Psychology, Klagenfurt, Austria.

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

[26] 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.

[27] Nick Parlante. 2024. CodingBat. Online Publication. https://codingbat.com/about.html

[28] Andrew Petersen, Michelle Craig, Jennifer Campbell, and Anya Tafliovich. 2016. Revisiting why students drop CS1. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. 71–80.

[29] Thomas W. Price, Yihuan Dong, Rui Zhi, Benjamin Paaßen, Nicholas Lytle, Veronica Cateté, and Tiffany Barnes. 2019. A comparison of the quality of data-driven programming hint generation algorithms. *International Journal of Artificial Intelligence in Education* 29, 3 (2019).

[30] Rajendra Raj, Mihaela Sabin, John Impagliazzo, David Bowers, Mats Daniels, Felienne Hermans, Natalie Kiesler, Amruth N. Kumar, Bonnie MacKellar, Renée McCauley, Syed Waqar Nabi, and Michael Oudshoorn. 2021. Professional Competencies in Computing Education: Pedagogies and Assessment. In *Proceedings of the 2021 Working Group Reports on Innovation and Technology in Computer Science Education* (Virtual Event, Germany) *(ITiCSE-WGR '21)*. ACM, New York, 133–161. https://doi.org/10.1145/3502870.3506570

[31] Lianne Roest, Hieke Keuning, and Johan Jeuring. 2023. Next-Step Hint Generation for Introductory Programming Using Large Language Models. arXiv:2312.10055 [cs.CY]

[32] Valerie J. Shute. 2008. Focus on formative feedback. *Review of Educational Research* 78, 1 (2008).

[33] Kurt VanLehn. 2011. The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems. *Educ. Psych.* 46, 4 (2011).