PATT43: The 43rd International Pupils' Attitudes Towards Technology Research Conference
Hosted by the Swedish National Centre for School Technology Education (Cetis) and Technology and Science Education Research (TekNaD), Linköping University, Norrköping, Sweden, June 15th to June 18th, 2026

The Problem Space: A Tool for Teaching and Learning Programming

Lennart Rolandsson, Department of Education Uppsala University
lennart.rolandsson@edu.uu.se

Abstract

Studies show that novice programmer use a range of self-made strategies when problem solving. They step through code, insert print statements, and test incrementally. Yet their debugging skills are often unsystematic. This is partly due to cognitive overload and reliance on habitual practices. Teachers also struggle to interpret students' questions when prior actions remain hidden in what has been written in the code. Consequently, teachers require a didactical tool that support the reconstruction and interpretation of what made the students' question important in the first place. In the paper, we draw from *pragmatism* and a student case where novice programmers are solving syntactical, contextual and semantical problems simultaneously, to argue for a *problem space*. The concept serves as a didactic tool to support teaching and learning problem-solving in programming, as teacher and students become aware of the different problems, when the problematic situations translate into inquiries. The concept does so without disclosing the solutions to students' problems and opportunities to gain valuable insights.

Key Words: Programming, Teaching Instructions, Problem-Solving, Problematic Situations, Pragmatism

1 INTRODUCTION

In upper-secondary programming courses, teachers spend a great deal of time teaching the syntax of a programming language—that is, the language's formal rules—because syntax errors are the easiest to identify and correct. Today, with artificial intelligence (AI), that kind of instruction is redundant, since tasks can be solved with a chatbot (Jonsson & Tholander, 2022): it writes the code, interprets the code, and explains the code. There are also studies (Robins et al., 2003; Grover & Pea, 2013) showing that learners of programming must be offered a meaningful context in order to develop these skills, and therefore programming education needs to be carefully designed, since AI can generate code with little effort, while students have to thoroughly understand, and use the basic concepts in computer science (e.g., variable). The present paper addresses the intersection of AI in programming education and the didactical imperative for meaningful learning experiences in pragmatism, both of which are developed more fully in the subsequent subsections.

1.1 A problem space for teaching and learning about problem-solving

According to researchers (Murphy et al., 2008), it does not matter very much which debugging strategies one teaches in programming education, since students typically debug randomly regardless of the error and use different methods as they occur to them. They argue that this behavior stems from students' lack of ability to recognize when their cognitive capacity is overloaded. In addition, they tend to remain in established habits. Instead, the researchers suggest students should ask themselves reflective questions: "What else could I try?", "Is this too much to keep track of in my head?", and "What are other possible sources of the bug?". However, learning and teaching programming can be dynamic, especially as many of the programmer at upper-secondary are novices and not reasoning logically with computer science concepts, making it difficult for teachers to discern how far students have progressed from a problematic situation toward an understanding of how to tackle the problem. Additionally, we find these reflective questions challenging to formulate as a novice, when occupied with multiple problems.

In the following, we propose a didactic tool, *problem space* as a shared reference of understanding, to unpack problematic situations while depicting the complexity of how one or many problems are co-existent in problem-solving. The notion of a *problem space* was established by Middleton (2002), where a chess player moves from an initial configuration (the problem state), explores a solution space or search space containing multiple possible continuations, and ultimately reaches a resolution (the goal state). As Middleton notes, a problem space may permit numerous moves, yet only a subset of these will be about problem-solving. Over time, progress is achieved by selecting the most appropriate move within the search space; importantly, what counts as the "best" move is not value-neutral but shaped by contextual norms and priorities. In the paper, we expand on Middleton's problem space making it suitable for the teaching of novice programmers when learning about problem-solving.

1.2 The theoretical and analytical framework

The language and discourse about problems when programming is deeply embedded in teachers' and students' actions when problem-solving, often framed as inevitable or beneficial — and it is precisely this naturalization of concepts that Wittgenstein addresses in his later work (1953). Therefore, in the following we *deconstruct* and *reconstruct* what is naturalized in students' problem-solving processes, and made to appear among us all as inevitable or universal. However, problems are in fact contingent and particular in many ways, and we would like to pinpoint that with a problem space.

In fact, Wittgenstein argued that words in a language game do not need to be clearly defined to be meaningful (Jago, 2007), a statement that is crucial when coding and discovering the logical solution when programming. According to Pinto (2024), the later Wittgenstein conceived language based on its different uses, in a language game, abandoning the essence of language and the limits imposed by the logical world. Instead, he suggested an account of otherness (Zarębski, 2023) shifting perspective from finding the one right answer to understanding the differences behind how students arrive at various conclusions. In the paper reconstruction is an act of

expanding our understanding problem-solving processes, creating space for non-expected language games, recognizing that the person who speaks differently is not speaking *wrongly*, but playing a different game, one with its own internal coherence and dignity. After all, many of Wittgenstein's examples are from his teaching experiences and he pursued that logical thinking is not always about words and symbols (Maruyama, 2000; Stickney, 2017). In the following we draw from these principles and ideas of otherness and language game, to unfold the very process of learning when programming.

For the deconstruction, we deploy an analytical framework (Östman et al., 2019) to move beyond our prior assumptions and interpretations of students' learning when programming. It is a framework that goes into detail when students deepen their understanding of the problems with multiple possible paths to move forward. Researchers (Östman & Wickman, 2014; Östman & Öhman, 2022) argue that students have an opportunity either to deepen old habits or to acquire new habits as a result of the experiences they gain when they choose to engage with in the problematic situation and initiate an inquiry. In this text, we use this analytical framework to unravel students' actions when *encountering*, *standing fast*, creating *gap*, and *relations*. Four actions that describe, in detail, what students draw on in their immediate surroundings, as they work through and make sense of what is actually at the core of their communication. The first action (*encountering*) concerns the meeting with something new or unexpected; the second (*standing fast*) concerns prior knowledge; the third (creating a *gap*) concerns the uncertainty they display when they hesitate and ask questions to themselves and others; and the fourth (creating *relations*) concerns the meaning they create in relation to one another, the task description, the programming environment, the structure of the code, and what they already know about the code. In addition, students are encountering code instructions, the computer's error messages, and one another's utterances in an ongoing conversation, in order to create relations to what is happening, something that we consider to be transactional (Dewey & Bentley, 1949) by nature.

Central when analyzing the transcripts are Dewey's ideas about the learning that emerges when "the living creature undergoes, suffers, the consequences of its own behavior" (Dewey, 1920, p. 129) — precisely what happens when students are encountering a problematic situation, initiate an inquiry, and resolve the problems that arise in the inquiry process. The latter — resolving the problems — can be observed in whether students gain a valuable experience or insight: what Dewey describes as a consummatory experience (valuable insight).

2 METHODOLOGY

The paper is grounded in a relational and transactional perspective when learning. In this chapter we describe the specific task and the importance of students' environment, focusing on students' relationships with their immediate surroundings—such as the computer, keyboard, screen, and peers — and how a problematic situation can be analysed with the theoretical framework described above.

2.1 The setting, the data collection and the analysis

To support our research, we situated our study in a setting and a task that requires students to draw on their previous experiences of problem-solving. To carry out the data collection, we used a screen-recording tool to capture students' actions, including mouse movements, keystrokes, and the conversation between the students S1, S2 and computer (C). The two students are novice programmer, and we twisted the task somewhat to unravel how they decide to move forward when the context is not clear to them. The task given to them was as follows:

```
1 # Pelle has been given a few lines of code; he doesn't understand
what the program is supposed to do.
2 # Can you explain the program to Pelle and then make sure it
works as intended
3
4   user_input = input("Knorglf florpa slorf: ")
5   ADJUSTMENT = (9/5)
6   OFFSET = 32
7   result = ADJUSTMENT * user_input + OFFSET
8   print("Blorflengor ", user_input, "p å florf elg: ", result)
```

The code, in the task, will produce a common syntax error in line 7: a text string (user_input) is multiplied by a decimal number (ADJUSTMENT). Therefore, the setting is prepared with something that become problematic in different ways, among the students (See chapter 3).

In the following chapter, students' actions have been *deconstructed* as far as we consider necessary in order to *reconstruct* a narrative. In practice, this means that the author of the paper worked closely together with the upper-secondary teachers (acknowledged) in a strictly empirical manner to unravel the problematic situations in the transcripts, with many cycles, so that students' actions became visible and the narrative became trustworthy. Initially, we searched for the gaps and insights (See Chapter 1.2), when students used signs, actions (the mouse cursor to focus and communicate), the computer environment and error messages to revise the code. Later on, we searched for how the students conducted the language game. Some situations (next chapter) may initially appear unproblematic, but the analysis unravels how students' questions, hesitations and insights are analytical windows for us to study what is actually problematic, and how they decided to initiate inquiry. For that reason, you as a reader may need to examine the whole transcript before going into the details.

2.2 Disclosure Statement

Generative AI tools were employed solely for assistance with translation and phrasing

3 RESULTS AND ANALYSIS

In the following transcripts we explore how the problematic situations evolve and the two students (S1 and S2) initiate inquiry with two types of problems (syntax and context, Table 3.1 and 3.2), and end with a third type of problem, a semantic problem[i] (Table 3.3).

Table 3.1

The students move into two types of problems (syntax and context)

Line	Agent	Utterance	Action
121	S1	I still don't know what "knorg florpa slorf" is.	
122	S2	But it doesn't matter what it is.	gestures
123	S2	It's some kind of question.	
124	S1	They *are* asking.	
125	S2	They're asking for something.	
126	S1	They must… yeah	talk over each other
127	S2	We don't give a damn what they're asking.	gestures
127b	S1	—	laughs and leans back
128	S2	Ignore it.	
129	S2	With the answer they get, they're supposed to do something.	points at the screen
…	…	…	…
132	S2	What do they want to do?	
133	S2	Adjustment multiplied —	
133b	S1	—	moves closer to the screen
134	S2	User input times.	
135	S2	Plus offset.	
136	S1	What?	grins and looks down at the table
137	S2	I'm trying to put it into a context and see if it makes sense here, but…	rubs his eyes
138	S1	Noo—	

139	S2	Adjustment times user_input plus offset	reads line 7 silently; hands in hair
140	S2	Is user input supposed to be a number?	
141	S2	No, it's still a string.	
143	S1	A florf is in there too.	laughs; puts a hand to his forehead; mutters
144	S1	It's florf and slorf.	
145	S1	No, God—my brain.	
146	S1	I feel like I'm overthinking…	
147	S2	I think you're focusing on the wrong thing.	grins
148	S1	—	laughs
150	S1	I'm really confused	

Deconstruction: A gap emerges: S1 does not understand "…what Knorg florpa slorf is" (121). S2 tries to fill the gap with, "But it doesn't matter what it is … it's some kind of question … they're asking for something" (122, 123, and 125). S1 notes that "they are asking" (124). S2 says, "We don't give a damn what they're asking. Ignore it" (127 and 128). S1 encounter S2's statement with laughs as he leans back. S2 relates to the code as he continues: "With the answer they get, they're supposed to do something" (129). Then S2 opens a new gap with another question: "What do they want to do?" (132). S1 leans forward (133) and listens as S2 reads aloud (132 to 135). S1 encounter S2 with a question as the gap is deepened: "What?" (136). S2 explains what he recently did: "I'm trying to put it into a context and see if it makes sense" (137), and we witness in real-time how they are using words and actions to bridge a gap. S2 then reads line 7 silently—"Adjustment times user input plus offset" (139) —and another gap is opened with a new question: "Is user input supposed to be a number?" (140). S2 immediately respond to his own question and fills the gap with, "No, it's still a string" (141). At the same time, S1 works with the initial gap: "A florf is in there too … I feel like I'm overthinking … I'm really confused" (143, 146, and 150). S2 thinks S1 is "focusing on the wrong thing" (147), and we witness how they are aware of their differences.

Reconstruction: The students express uncertainty about "something" in a language game. S2 argues that they should "not give a damn what they're asking" (127) because they know the code is supposed to "do something" (129) with the input. A problematic situation emerges, and the deconstructive analysis reveals that the students are, in fact, engaged with different problems. Their divergent orientations become particularly visible when S2 encounters difficulties with *user_input* (line 4 and 7), where two different data types are in play, while S1 tries to understand the meaning of "Knorg florpa slorf" on line 4, which contributes to the student's confusion (136, 143, 146, and 150). They initiate the inquiry with different problems in mind.

Table 3.2

The context problem is solved

Line	Agent	Utterance	Action
152	S1	But are they trying to ask, like, in some way, how—how much you want… how many of those adjustment-things you want to do?	
153	S2	How many adjustments you want to do?	
154	S1	That that's kind of what they're trying to ask for.	points at the screen
155	S1	So that you have…	takes the mouse and closes the terminal window
156	S1	—	execution for the 3rd time
156b	C	—	prints: "Knorglf knorpa slorf"
157	S1	Because you *will* get a result.	places the mouse cursor in the terminal window
158	S2	No…	
159	S1	But, hypothetically.	gestures broadly with both arms
160	S2	You *do* want a result.	emphasize "result", while taking the keyboard to type
161	S1	Exactly.	laughing
162	S1	But— so, yeah—	
163	S2	But if we type a number here now, it's still a string.	leans forward, Submits 6
164	S1	But it won't work.	Puts his hand to his forehead
165	C	—	computer displays an error message: "TypeError: Can't multiply sequence by non-int of type 'float'"
166	S1	Exactly! … but—	

Deconstruction: S2 opens a gap as he hesitates: "how many of those adjustment-things do you want to do?" (153). In order to fill the gap, S1 points to line 4 and reply, "That's kind of what they're trying to ask for," (154). S1 takes the mouse and keyboard to run the code (156). They encounter the computer's prompt on the screen together: "Knorglf knorpa slorf." The students have different views of the "result": S1 says, "Because you *will* get a result … hypothetically" (157 and 159) and S2 states with emphasis "No … you *do* want a result" (158 and 160), something S1 agree with "Exactly" (161) while laughing. S2 moves closer to the screen submitting the number 6 (163). S1 puts his hand to his forehead. The computer (C) returns an error message (165), and S1 reply "Exactly!" (166).

Reconstruction: As the transaction is explicit, we can see how "result" is still not decided in the language game, but takes shape through the participants' overlapping actions (including the computer) in the course of the inquiry. S2 submit a number, and S1 closes the transaction with his final "Exactly!" (166) — an expression suggesting that S1 gained a valuable insight (*consummatory experience*) because of the error message (165), and the word "hypothetically" (159) is used to move forward, and they finally agree on how the "result" could be used to communicate different things (the code outcome and result on line 7), in the inquiry.

Table 3.3

The syntax problem is solved and semantic problem emerges

Line	Agent	Utterance	Action
175	S1	The reason why it is there	adds a parenthesis on line 7 around user_input
176	S1	You want to multiply, with that, and then it must be an integer	adds an "int" before the parenthesis
177	S2	—	runs the program for the 4th time
178	S1	But then you also need something that says it has to be written in numbers, too.	hides face in both hands
179	S2	—	Submit 7
180	C	—	program runs; prints: "Blorfengor 7 p å florf elg: 44.6"
181	S2	I'm thinking that's what "knorgl glorpa slorf" means.	points at the screen (the output) and laughs
182	S1	Mmm.	
183	S2	Okay, then we get [a] number.	

184	S1	Yes.	
185	S2	Blorfengor seven. p å florf elg, 44.6	gazes upward toward the ceiling and rolls his eyes

Deconstruction: S1 explains the reasons why he revises the code in line 7 (175 and 176), to `result = ADJUSTMENT * int(user_input) + OFFSET`. The code is a profound statement, in the inquiry, as they execute the code for the fourth time (177). S1 says, "But then you also need something that says it has to be written in numbers" (178). S2 submit the number 7 (179). The computer responds and they encounter "Blorfengor 7 p à florf elg: 44.6" (180) on the screen, and S2 expresses a feeling by laughing. S2 says, "Okay, then we get [a] number" (183), and S1 responds, "Yes" (184). The S2 gazes upward toward the ceiling while rolling his eyes (186).

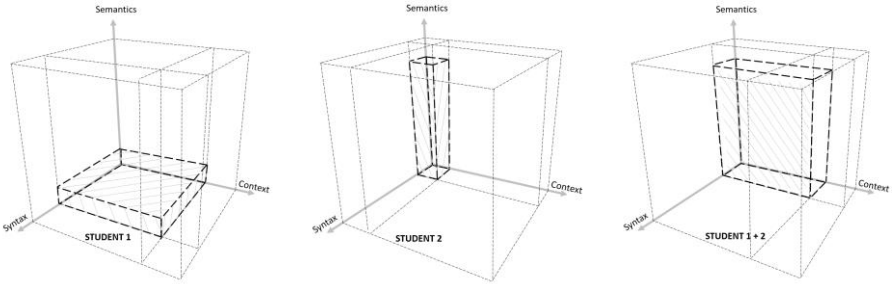
Reconstruction: S1 changes the data type from string to integer and executes the code. The previous error message is no more, and we understand a bit more of S1's insight (see previous transcript) and the reasons for correcting the code. S2 does not comment on that change, and we can therefore infer that knowledge of different data types *stands fast* to them both. In other words, the students have done similar things (switching data types) before, in order to continue and move forward in the inquiry. However, the informational texts (on lines 4 and 8) are at foci while expressing "something that says [to the user] it has to be written in numbers" (178) and we can tell they are working within the dimension of semantics, to understand what the code actually imply for a user. S2 goes from laughter to eye-rolling in a couple of seconds, telling us he is frustrated.

3.1 Summary

In the three transcripts describe above, the students' responded differently when problems appear, i.e., a syntax error planted in the code, did not imply a syntax- or context problem in any straightforward sense. This is described eloquently in the first transcript where one of the students (S1) experience a cognitive overloaded, and the other student (S2) want to move forward. This is because of different problem spaces (Figure 1a and 1b respectively). However, in the third and the last transcript, the students expanded, focusing on the semantic problem, and we can imagine how the syntactical problem is solved. In short, the two students enter into one or more problems simultaneously trying to solve the problematic situation. The two students worked in two dimensional spaces — initially syntax in relation to context (Figure 1a) and later semantics in relation to context (Figure 1c). In contrast with the rest of our data, we found no problematic situation in which students oriented exclusively to the semantics-syntax dimensions. One possible explanation for this absence is that students were typically able to easily detect syntactic errors, thus preventing the relevant situations from occurring in our data. Addressing such inquiries would most likely require students to engage in more complex structures.

Figure 1a, 1b and 1c

The problem spaces experienced by the two students.



4 DISCUSSION

There are computer scientists arguing that "coding is to programming what typing is to writing" (Lamport, 2015), meaning the code serves as a tool for thinking, and Hazzan et al. (2026) suggest alternative conceptions as a way for teaching problem-solving in programming, moving away from the discourse of misconceptions where one perspective is the standard and everything else is a "mistake". However, our upper-secondary students are novice programmer, and there is a distinction between what computer scientists and novice programmer think, and what the computer interprets, a distinction that emerges as the code is malfunctioning, and we are interested in what novice programmers are actually thinking when problematic situations occur and transform into inquiries. This distinction proved significant in our data, where a considerable number of novice programmers continued their inquiry beyond the point at which the problematic situation had been resolved, indicating that the sense of what counted as a problem was not fixed in advance but was continuously negotiated in use. In short, we are interested in the students' language game (presented by Wittgenstein) and how novice programmers are negotiating alternative courses of action (not alternative conceptions) to move forward, while teachers need to preserve students' prerogative to remain in charge of their work. In short, a problematic situation is commonly a complex web of problems and with a better understanding of the existent problems among students, we as teachers can assist them better than AI. Besides many of our students, as novice programmers talk about "this" and "down there" etc., instead of using the names of the computer science structures involved.[ii] This is also a language game that most likely stems from poor experiences or missing opportunities of having a valuable insight. A contributing source of this lack of experiences may be located in teachers' lack of clarity concerning both the appropriate timing and approach to offer students the insights. In response, we propose the problem space designed to render visible students' original epistemic values in the thinking process, as a complementary concept to the prevailing emphasis on (mis)conceptions, strategies, and industrially inflected values — such as efficiency and logical reasoning — that characterize many programming practices today.

5 ACKNOWLEDGMENTS

This paper builds on a practice-based research project in collaboration with Jens Andreasson, Stefan Folkesson, Martin Lundqvist, and Gustav Wiberg, all of whom are teachers in programming at upper secondary schools.

6 REFERENCES

Dewey, J. (1920). Reconstruction in philosophy. In: Jo Ann Boydston (ed.), *The Middle Works (1899–1924), Volume 12*. Carbondale and Edwardsville: Southern Illinois University Press.

Dewey, J. & Bentley, A. F. (1949). *Knowing and the Known* (1st ed.). Boston: Beacon Press.

Grover, S. & Pea, R. (2013). Computational Thinking in K–12: A Review of the State of the Field. *Educational Researcher 42*(1), 38-43.

Hazzan, O., Ragonis, N., & Lapidot, T. (2026). Learners' Alternative Conceptions in Computer Science. In: Guide to Teaching Computer Science. Springer, Cham.

Jago, M. (2007). Wittgenstein. Humanities-Ebooks.

Jonsson M. & Tholander J. (2022). Cracking the code: Co-coding with ai in creative programming education. In: Proceedings of the 14th Conference on Creativity and Cognition. CC '22, 5-14. Association for Computing Machinery, New York, NY, USA.

Lamport, L. (2015). Magazine Q. The man who revolutionized computer science with math. https://www.youtube.com/watch?v=rkZzg7Vowao.

Maruyama, Y. (2000). Wittgenstein on Teaching and Otherness: toward an Ethic of Teaching (a PhD Dissertation). Tallahassee, FL: The Florida State University.

Murphy, L., Lewandowski, G., McCauley, R., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: the good, the bad, and the quirky -- a qualitative analysis of novices' strategies. *SIGCSE Bull. 40*(1), 163–167.

Pinto, A. O. (2024). Language and Thought in Wittgenstein. *Advances in Social Sciences Research Journal, 11*(12), 14-23.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education, 13*(2), 137-173.

Stickney, J. (2017). Wittgenstein as Educator. In: Peters, M., Stickney, J. (eds) *A Companion to Wittgenstein on Education*. Springer, Singapore.

Wittgenstein, L. (1953). *Philosophical Investigations*. Blackwell.

Zarębski, T. (2023). Fictitious Language Games, Otherness, and Philosophy of Education: A View on the Later Wittgenstein. *Studies in Philosophy and Education,* 42, 323-336.

Östman, L., van Poeck, K., & Öhman, J. (2019). A transactional theory on sustainability teaching: Teacher moves. In: Van Poeck, K., Östman, L., Öhman, J. (Eds.) *Sustainable Development Teaching: Ethical and Political Challenges*. New York: Routledge, 140-152.

Östman, L. & Wickman, P.-O. (2014). A pragmatic approach on epistemology, teaching, and learning. *Science Education, 98*(3), 375–382.

Östman, L. & Öhman, J. (2022). A transactional methodology for analysing learning. *Mind, culture and activity*, 1-17.

[i] Context refers to what the task is asking and how students understand it. Syntax refers to the rules and structure of the code that allow the computer to read and run it. Semantics refers to what the code actually *means* – what happens and why the computer executes it.

[ii] It is extremely hard for a student to recount a sequence of events, when the s/he is a novice, missing the conceptual understanding. Students' narratives easily become a reconstruction after the fact—based on the outcome rather than on the choices available when the problematic situation initially appeared.