Codebook

Interpret the prompts "at face value," minimizing the need to infer additional context or information.

Programming Behaviors

Key distinction between speculation about code vs actually written code.

PIN: Interpret prompt

Statements about or demonstrating interpreting or questioning the prompt, reconsidering actions in reference to the prompt, or decomposing the problem into goals requirements and or sub-problems.

- Mention of question prompt
    - Random nouns that have nothing to do with programming relate to prompt
        - E.g. snowing, employees
        - EXCLUDES common programming terms (including numbers, numerical operations)
- Mention of goals or actions with *explicit* connection or relation to question prompt
    - If no explicit mention of prompt, see SPL
- Awareness of prompt or inference about problem prompt may be required.

PSA: Search for analogous problems

Statements demonstrating intent to reuse knowledge or code from related problems

- Relating current problem or task to a prior one
    - (e.g. "problem looks like some sort of countdown")

PSS: Search for solutions

With some understanding of a problem, programmers seek solutions that will solve the problem by adapting solutions to related problems or by finding solutions in textbooks, online, or from classmates or teachers. During solution search, learners may monitor the extent to which they have searched and the degree to which the search was satisfactory.

- use of other resources (e.g. notes, other websites such as Python Tutor)
- Consideration of using a potential code construct (speculative)
    - E.g. "I thought about for loops"

    - E.g. "I could use…" or "I originally considered using a list…"
- Not just about acknowledging knowledge or awareness (see SCM)

PEP: Evaluate a potential solution

Statements of or demonstrating testing or evaluating outcomes intent to test a solution or identifying why code was (not) meeting expectations.

- Evaluation of idea or potential implementation relative to criteria (speculative)
    - Not just describing what a construct does.
    - Considering what outcome or why the idea works
- Considerations of how to test code
- NOT evaluation of written code (see PEI)
- NOT evaluation of test pre-defined test cases (see PEI)

PIM: Implement a solution

With a solution in mind, programmers must translate the solution into code using their programming languages and tools. Learners may self-regulate their awareness of working memory limitations and manage prospective memory for future tasks.

- description of code implementation (very common)
    - Mentions of code constructs in present (use, am using), present-participle (using), or past tense
        - If future, see SPL ("I will use…", "I planned to use…")
        - If speculative, see PSS ("I thought about…", "I could use…")
- Code has been written already, retrospective + summative
    - Talking about code construct in past tense
- Includes what their implementation may be missing
    - E.g. "I did not have a return statement"

PEI: Evaluating an implemented solution

After implementing a solution, programmers iteratively converge toward correctness, evaluating how well their implementation solves the problem, usually by testing and debugging. Learners may self-regulate their certainty in an implementation's correctness to prevent overconfidence.

- Mentions of testing with predefined test cases
- Mentions of implementation requiring something after at least some implementation
    - E.g. "I tested my code then realized I needed another if statement"
    - NOT realizing a requirement prior to actual implementation (see SCM)
        - E.g. "I knew I would need a for loop"

- Mentions of correctness
    - E.g. “i wrote this thing, realized it wasn’t passing a test, and so i changed it”
    - Includes mentions of complexity or efficiency

Self-regulation

SPL: Planning

Statements of intended work goals or intended order of work

- Mentions of planning of tasks
- NOT progress towards goals (see SPM)
- Future-oriented relative writing code, NOT retrospective or speculative
    - Focus on intention to do something
    - NOT about code that was written (see PIM)
    - NOT about considering a solution (see PSS)
- Explicit mentions of writing pseudocode
- Exception: grammatical error related to tense

SPM: Process Monitoring

Statements about work being started, identifying work currently in progress, when a task is complete, or statements that identify actions as part of their process.

- MULTIPLE steps towards goal or completion
    - A single step is insufficient (e.g. NOT "I first thought about for loops")
    - E.g. "first I defined my variables, *then* I …"
    - Chaining
    - OK if single word identifies multiple steps (e.g. "before," "after," "**then**")
- Mentions of going through multiple requirements or steps in prompt (also PIN)

SCM: Comprehension monitoring

Statements identifying known or unknown programming concepts or understanding of the problem prompt. Reflection about the understanding of code or problem prompts.

- EXPLICIT mention of what they know or do not know
    - E.g. "I *know* I had to use loops"
    - NOT implicit statements of comprehension
        - "I could use loops"
- Includes forgetting something in code or forgeting concept

    - E.g. "I forgot to add a colon" or "I forgot how to nest loops"

SRE: Reflection on cognition

Statements reflecting on prior thoughts, behaviors, or feelings.

- Mentions of their own cognition, thoughts, and memory
    - E.g. assuming or believing something, biases, feelings
    - E.g. "I did [action]. Then I realized that I did not have to do that" (would also be SPM, PEI)
- NOT about reflections on goals (see SPL)
- NOT about reflecting on they know or do not know (see SCM)
- NOT about reasoning (see SSE)

SSE: Self-explanation/ rationale

Statements of code explanation for increased understanding or to provide rationale to decisions or behaviors.

- Explanation of why implementation (e.g. code) performs a certain way
    - Includes explanation of bug or error
    - explanation should contribute actual substance
        - If explanation too vague, then don't code
            - E.g. "I could use a list because they are useful" (not SSE)
        - OK for explanation to be wrong
            - E.g. "I could use a list because lists are strings" (is SSE)
- Justification or explanation of actions or behaviors
    -
    - E.g. "I did [action]. Then I realized that I did not have to do that <u>because …</u>" (would also be SPM, SRE)
    - “I knew you could multiply an integer by a string, so i did that”
    - “I want to compare current character with….”
- NOT just description of code (see PIM)
- NOT evaluation of code w/o explanation (see PEP, PEI)

N/A: No codes applicable

Only selected when no other code is applicable

Examples

At first I tried to make a separate list of the numbers [PIM] in int form but I realized I did not have to do that [SRE, PEI] so [SPM] I changed it. There may be a way to do this without separating the given string into 2 strings[PSS] but I don't know what that would be[SCM].

- SRE, PSS, PIM, PEI, SPM, SCM

Notes

Prior def'n and codebooks

Programming Behaviors	Definition
Interpret prompt	Statements about or demonstrating interpreting or questioning the prompt reconsidering actions in reference to the prompt or decomposing the problem into goals requirements and or sub-problems.
Search for analogous problems	Statements about or demonstrating intent to use code they have previously written or use of examples from outside sources.
Adapt a solution	Statements of or demonstrating changing or refining code.
Evaluate	Statements of or demonstrating testing or evaluating outcomes intent to test a solution or identifying why code was not meeting expectations.
Self-Regulation	**Definition**
Planning	Statements of intended work goals or requirements an intended order of work
Process Monitoring	Statements of start times stop times duration of coding session about work being started identifying work currently in progress when a task is complete or statements that identify actions as part of their process.
Comprehension monitoring	Statements identifying known or unknown concepts or solutions.
Self-explanation	Statements of code explanation for increased understanding.
Reflection	Statements reflecting on prior thoughts of behaviors.
Rationale	Statements that provided rational to decisions or behaviors.

Table 1: Programming and self-regulation behaviors (from prior work [11]) coded in the journals, with definitions.

The problem solving stages we propose to teach include six stages that prior literature on the psychology of programming suggests are essential to successful programming. While nominally sequential, the stages are re-visited frequently as programmers iteratively implement a solution and discover knowledge about the problem and solution that was not initially apparent. The stages are:

- *Reinterpret problem prompt.* Programming tasks typically begin with some description of a problem, which programmers must understand, interpret, and clarify. As with other forms of problem solving, this understanding is a cognitive representation of the problem used to organize one's "continuing work" [23]. The more explicit this interpretation process, the more likely a programmer will overcome ambiguities in the problem [42].
- *Search for analogous problems*. Programmers draw upon problems they have encountered in the past, either in past programming efforts or perhaps in algorithmic activities they have encountered in life (e.g., sorting a stack of books or searching for one's name in a list) [25]. By reusing knowledge of related problems, programmers can better conceptualize a problem's computational nuances.
- *Search for solutions*. With some understanding of a problem, programmers seek solutions that will satisfactorily solve the problem by adapting solutions they have used in the past or by finding solutions in textbooks, online, or from classmates or teachers [9,29].
- *Evaluate a potential solution*. With a solution in mind, programmers must evaluate how well this solution will address the problem. This includes actions like feasibility assessments, mental algorithm simulations, or other techniques of sketching or prototyping a solution before implementing it [31].
- *Implement a solution.* With an acceptable solution in mind, programmers must translate the solution into source code using their knowledge of languages and tools.
- *Evaluate implemented solution*. After implementing a solution, programmers iteratively converge toward a solution by evaluating how well their current implementation solves the problem. This typically involves software testing and debugging [29,42].

Other notes

"I noticed this. And then I did this…"

Based on Loksa et al SIGCSE 2020, Loksa et al CHI 2016.

Data from Loksa et al SIGCSE 2020:

Behavior	#1	#2	#3	#4
Interpret	0-1	0-1	0-1	0-1
Search	0-7	0-2	0-2	0-3
Adapt	0-6	0-4	0-1	0-1
Evaluate	0-11	0-6	0-8	0-14
Planning	0-19	0-9	0-19	0-7
Process Monitoring	0-25	0-25	0-54	0-43
Comprehension monitoring	0-6	0-2	0-4	0-2
Self-explanation	0-24	0-17	0-13	0-9
Reflection	0-13	0-7	0-7	0-8
Rationale	0-17	0-8	0-8	0-7

Table 3: The range of student entries exhibiting each self-regulation or programming behaviors, by assignment, showing higher frequencies of evaluation, planning, process monitoring, and self-explanation than other behaviors and interpret only occurring once per assignment.

Code	High (12)	Moderate (12)	Low (7)	Total
Process	100%	100%	100%	100%
Evaluate	100%	100%	100%	100%
Search	100%	100%	86%	97%
Reflection	100%	100%	43%	87%
Explanation	100%	100%	29%	84%
Rationale	92%	100%	29%	81%
Planning	92%	75%	29%	71%
Comprehension	100%	33%	29%	58%
Interpret	75%	8%	0%	32%
Adapt	75%	8%	0%	32%

Table 4: The three clusters of behavior (and size of cluster). Each percentage indicates the proportion of students in the cluster who exhibited the behavior at least once in a journal.

who demonstrated *mature* reflection were those who identified indi-

- Mimic table on the right; instead of high/mod/low, compare PS1 vs. PS3

Graveyard

Programming behaviors

Adapt a solution

1. Statements of or demonstrating changing or refining code.

Self-Regulation

PSS: old definition
Statements about or demonstrating intent to use code they have previously written or use of examples from outside sources.

Programmers draw upon problems they have encountered in the past, either in past programming efforts or perhaps in algorithmic activities they have encountered in life (e.g., sorting a stack of books or searching for one's name in a list) [25]. By reusing knowledge of related problems, programmers can better conceptualize a problem's computational nuances.

SRA: Rationale

Statements that provided rationale to decisions or behaviors.

- Justification or explanation of actions or behaviors
    - Not just description of actions or behaviors (see SRE)
    - E.g. "I did [action]. Then I realized that I did not have to do that <u>because …</u>" (would also be SPM)
- NOT about explanation of implementation or code (see SSE)