

Beyond Syntax and Tracing: A Multidimensional Diagnostic Framework for Novice Code Comprehension

Valéria Cavalcanti^{1,2}, Wilkerson L. Andrade²

¹Academic Unit of Informatics – Federal Institute of Paraíba (IFPB)
João Pessoa – PB – Brazil

²Systems and Computing Department – Federal University of Campina Grande (UFCG)
Campina Grande – PB – Brazil

valeria.cavalcanti@ifpb.edu.br, wilkerson@computacao.ufcg.edu.br

Abstract. *Understanding code is a major challenge for beginning programming students, especially as Large Language Models (LLMs) have made code generation easier than understanding. Despite the availability of various assessment instruments, many approaches still analyze this skill through isolated tasks and results based solely on correct answers or grades. To address this problem, this work proposes the Three-Dimensional Comprehension (3DC) model, a multi-dimensional approach to diagnosing code comprehension. To operationalize the model, the LIFC process was developed, integrating multiple assessment instruments and tasks to collect and triangulate evidence generated by students during program reading and interpretation activities. The proposal also defines the LOAD diagnostic profiles, which represent different comprehension patterns observed among beginning students. Preliminary results indicate that integrating multiple pieces of evidence enables the identification of specific difficulties that traditional assessments based solely on final performance cannot adequately capture. Thus, the 3DC model seeks to support more detailed diagnoses and more targeted pedagogical interventions in introductory programming education.*

1. Introduction

Although code comprehension is a historically recognized problem, the use of Large Language Models (LLMs) has introduced a new scenario that has widened the gap between producing code and effectively understanding it, as identified by Andersen-Kiel and Linos (2024).

Several teaching strategies are found in the literature to improve code comprehension. Among them, some gain greater relevance in this new scenario, with an emphasis on Zavala and Mendoza (2017), suggesting the importance of first mastering the reading and comprehension of programs before subsequently manipulating and writing them. These strategies provide a set of instruments and tasks that aim to measure and improve the reading and interpretation of the program.

The works Lister et al. (2004), Wyrich et al. (2023), and Cavalcanti and Andrade (2025) identify some of these instruments, which typically use tasks such as predicting program output, tracing execution, identifying errors, explaining code snippets, and debugging programs. Using multiple such tasks in the comprehension assessment process provides a more comprehensive view of the student, as stated by Feitelson (2021).

As a contribution, this work proposes the 3DC model, a multidimensional approach to analyzing code comprehension among novice students. To this end, the model uses the LIFC process, which combines different assessment instruments to identify comprehension profiles and diagnose the main difficulties presented by students.

Although LLMs have reduced barriers related to code generation and modification, understanding such codes appears to remain an unaddressed gap. This gap is reinforced by the analysis conducted by Lehtinen et al. (2023), in which 27% of the students were unable to answer simple questions about their own code developed.

This scenario reinforces the need to investigate approaches capable of encouraging and assessing comprehension in a deeper and more multidimensional way, especially in a context in which writing code is no longer necessarily evidence of understanding.

To address this gap, this study is guided by two research questions: “How can novice students’ code comprehension be modeled to represent different learning profiles?”, and “How can novice students’ code comprehension be diagnosed through multiple assessments?”.

The remainder of this paper is organized as follows. Section 2 presents the theoretical background on code comprehension and discusses related multidimensional perspectives. Section 3 introduces the Three-Dimensional Comprehension (3DC) model and its diagnostic profiles. Section 4 describes the LIFC process, which operationalizes the proposed model through a structured sequence of assessment activities. Section 5 presents the experimental methodology adopted in the study, while Section 6 discusses the preliminary results obtained from the application of the proposed approach. Finally, Section 7 presents the conclusions, limitations, and directions for future work. The artifacts produced in this study are available in a public repository¹.

This research was conducted according to CNS Resolution No. 510/2016 for research in the Humanities and Social Sciences and was approved by the Research Ethics Committee under CAAE No. 92096425.5.0000.5185.

2. Background

In the literature, several works support the idea that students should first learn to read code and then write new code. For example, Zavala and Mendoza (2017) empirically showed that students should sequentially master the reading and interpretation of programs, code manipulation, and, finally, code writing.

The systematic review presented by Wyrich et al. (2023) indexed 95 studies related to code comprehension since 1979, with about half of them (51.6%) published from 2015 onward. This review identified that no formal consensus has been established on the definition of “Code Comprehension”. In each work, the construct is defined on the basis of the respective metrics used to assess it. As a result, several experiments were found to measure comprehension in different ways, making it difficult to compare these studies.

However, most of the analyzed works (81%) require participants to “provide information about the code”, with emphasis on the study by Shneiderman and Mayer (1979),

¹https://github.com/valeriacavalcanti/sbie_2026

which presented a cognitive model that integrates semantic and syntactic aspects in programmers' tasks, serving as inspiration for other works that later emerged in this area.

The code is not an ordinary text and requires specific cognitive processes from the reader, involving the recognition of structures, behavior prediction, mental simulations, and the recognition of functional abstractions. Given this cognitive complexity, several studies have investigated how programmers understand code and which mental processes are involved in this activity. Pennington (1987) conducted this analysis and concluded that programmers first understand how instructions are executed and later, with greater effort, for example, during maintenance tasks, understand what the program does.

These studies contributed to consolidating the idea that program reading is not based solely on syntactic understanding of the language, but also on the ability to organize information across multiple levels of abstraction, integrating structural, semantic, and functional aspects during the reading and analysis of programs.

Based on the perspective of Feitelson (2021) that code comprehension should not be represented by a single skill, task, or assessment metric, and considering that each of these tasks generates evidence based on distinct cognitive aspects involved in reading and interpreting programs, multidimensional models emerge in this context to analyze code comprehension.

Among the multidimensional models found in the literature, the "Block Model" by Schulte (2008) stands out. This model maps the evidence produced by students into dimensions, namely "Text surface", "Program execution", and "Functions", while considering levels of granularity, namely "Atoms", "Blocks", "Relations", and "Macrostructure". The dimensions represent different ways of understanding the code, while the levels of granularity represent the size or complexity of the element being analyzed. In this context, comprehension no longer considers only performance in isolated tasks, but instead interprets multiple interdependent competencies evidenced by students during programming activities.

The "Block Model" was proposed to support the planning and analysis of programming education, organizing different perspectives for reading and analyzing programs. The model made a significant contribution to consolidating code comprehension as a multidimensional phenomenon, in addition to inspiring several studies in the area.

However, the model provides limited support for the individualized diagnosis of difficulties presented by novice students during program reading. Although it makes it possible to identify dimensions and levels of granularity involved in program reading, the "Block Model" does not define how to integrate evidence produced by multiple assessment instruments, nor how to transform this evidence into diagnostic profiles capable of representing students' specific difficulties.

In view of these limitations, this work is motivated by the need to gather evidence produced in different code reading, analysis, and interpretation activities, considering multiple cognitive competencies associated with code comprehension. From this perspective, the next section presents the 3DC model, proposed to integrate different dimensions of code comprehension and support the construction of more detailed diagnoses of novice students' difficulties.

3. 3DC Model

Grounded in works that introduced the cognitive dimensions explored in this research, such as Pennington (1987), in which code comprehension results from the construction of two models, namely the “Program Model” and the “Situation Model”; Schulte (2008), which introduced a multidimensional model for analyzing program comprehension; and Kallia (2023), which states that building a coherent mental model of code requires the levels of understanding “Problem Level”, “Program Level”, and “Their Relationship”.

Considering that there is no formal definition in the literature, this work defines code comprehension as: “the ability to interpret the structure of code, understand how it behaves during execution, and relate this behavior to the problem that seeks to solve”.

The “Three-Dimensional Comprehension” (3DC) model assesses competencies specifically related to the reading, interpretation, and analysis of code produced by novice programming students or by third parties. This model is defined as a multidimensional construct resulting from the integration of the following dimensions.

- **Syntactic:** Represents the recognition of the program’s textual structure. That is, the ability to recognize, interpret, and use the lexical and grammatical elements of a programming language correctly.
- **Control-Flow:** Represents the understanding of the program’s dynamic execution. That is, the ability to build a mental model of code execution by following the control flow, state evolution, branches, repetitions, function calls, and the effects of instructions on memory.
- **Functional:** Represents the understanding of the program’s functional purpose. That is, the ability to understand the problem, the functional objective, and the computational intent of a piece of code, relating it to the problem domain.

The cognitive competencies associated with each dimension serve as references for interpreting the evidence produced by the students during the programming activities. Based on this evidence, it is possible to identify the difficulties frequently presented by novice students.

This model adopts multiple instruments as complementary sources of evidence. Thus, code comprehension can be characterized in a broader and more multidimensional way, allowing different diagnostic profiles of code comprehension among novice students to be represented.

For example, a student may correctly recognize syntactic structures without being able to predict the program’s behavior during execution, whereas another may partially understand the functionality of the code while still showing difficulties in the detailed interpretation of specific structures.

To support more targeted pedagogical interventions, the 3DC model maps the following diagnostic profiles resulting from the intersections among the model’s dimensions.

- **Literal Textual Comprehension (L):** In this profile, the student understands the problem and correctly recognizes syntactic elements but reads the code in a linear and static way, without adequately constructing the Notional Machine [Wilson et al. 2024]. This limitation makes the debugging process almost intuitive or based on trial and error.

- **Operational Comprehension (O):** In this profile, the student understands the syntax of the code and is able to simulate its execution through tracing, but has difficulty relating the observed behavior to the domain or functional purpose of the problem. As a consequence, the student has difficulty recognizing patterns, reusing solutions, and adapting the same logic to similar problems.
- **Algorithmic Reasoning (A):** In this profile, the student is able to understand the algorithmic strategy used to solve the problem, recognizing relationships between computational structures and solution patterns. However, the student shows syntactic weaknesses that limit code maintenance, especially correction.
- **Deep Code Comprehension (D):** The student integrates the three domains and is able to explain, trace, modify, correct, and transfer the program logic to new contexts.

Figure 1 presents the intersections between the model dimensions and the associated LOAD diagnostic profiles.

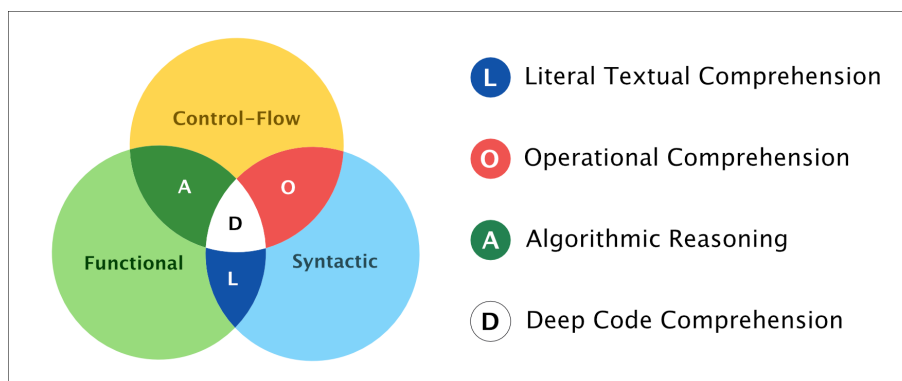


Figure 1. LOAD Diagnostic Profiles in the 3DC Model.

In summary, through the integration of instruments, the observed evidence is mapped through the competencies of each dimension. The 3DC model seeks to transform observations produced during programming activities into diagnostic profiles capable of supporting more targeted pedagogical interventions. Figure 2 presents an overview of 3DC.

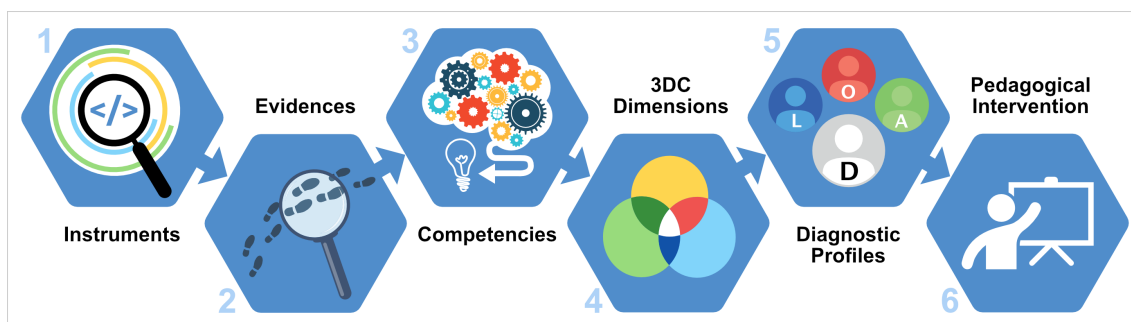


Figure 2. Overview of the 3DC Model.

In this context, assessment instruments are no longer used only for assigning grades, but instead function as sources of cognitive evidence about students' code comprehension.

Thus, rather than classifying comprehension only as “correct” or “incorrect”, or through a taxonomy, the 3DC model seeks to identify which aspects of comprehension are more developed and which require attention.

Once the 3DC Model has been presented, the question remains as to how this information can be observed and organized in practice. In this context, the next section presents the LIFC process, responsible for operationalizing the proposed model.

4. LIFC Process

This process is based on the premise that understanding code is not only about “knowing how to explain” (QLCs, [Lehtinen et al. 2023]) or “verbalizing” (Think-aloud, [Teague and Lister 2014]) what one is thinking, but rather being able to demonstrate such understanding through observable actions. In this context, the choice of instruments was guided by the principle of minimizing subjective judgments and the problems arising from communication in the natural language.

The methodology is organized into phases without feedback, such as whether the student was “right or wrong”. In this way, the learning effect during the analysis is eliminated. The idea is to measure the current state, not learning during the assessment process. Since students do not know how they performed, they move on to the next phases carrying their conceptions (correct or incorrect), thus avoiding the bias of demotivation.

The LIFC process (Leveling, Inference, Functionality, and Comparison) operationalizes the 3DC model through a structured protocol for collecting, organizing, and triangulating evidence produced by students during code reading, analysis, and modification activities. It is described as follows:

- **Leveling (L):** This first phase is inspired by the work of Lister et al. (2004), which investigated the level of programming knowledge and problem-solving ability through the application of a multiple-choice questionnaire. Similarly, it is also necessary to examine the hypothesis that the student may be “guessing” an alternative. In addition, it is necessary to verify whether the presence of the correct alternative acted as a trigger for the student to answer correctly;
- **Inference (I):** Unlike the previous phase, the objective here is to force the student to seek the answer without the visual support of alternatives, that is, only by reading the code. This phase was inspired by the work of Hannebauer et al. (2018), which presented a code with gaps that needed to be filled only by analyzing the context.
- **Functionality (F):** In this phase, it is necessary to connect a set of clues that determine the overall objective of the code. The intention is to assess the student’s ability to verify whether the behavior of the code satisfies the proposed problem, inspired by Banweer and Trytten (2024). If the student states that it does, they must perform a guided refactoring of the code. If the student states that it does not, they must present the corrected code. These maintenance tasks were inspired by the work of Szabo (2015).
- **Comparison (C):** In this phase, inspired by Izu and Mirolo (2020), the student must analyze a correct reference code, which solves the problem presented, together with other code pieces. For each code presented, the student must perform a comparative analysis to determine whether the code presented is functionally equivalent to the reference code.

As a way of observing specific cognitive competencies in a more controlled and interpretable manner, LIFC analyzes how students interpret basic computational structures, referred to as “Computational Patterns”, explored in introductory programming courses through different instruments and across different phases.

The “Pattern” is the object of analysis, and the same “Pattern” can generate different evidence depending on the task applied by the instrument. For example, the accumulator Pattern can be analyzed through tracing, debugging, code comparison, or test generation tasks, producing evidence related to the Syntactic, Control-Flow, and Functional dimensions of 3DC.

The choice of instruments to be used in each phase is essential for capturing the essence of the 3DC Model. Figure 3 shows the LIFC phases, with emphasis on the “Accumulator” Pattern used as an example.

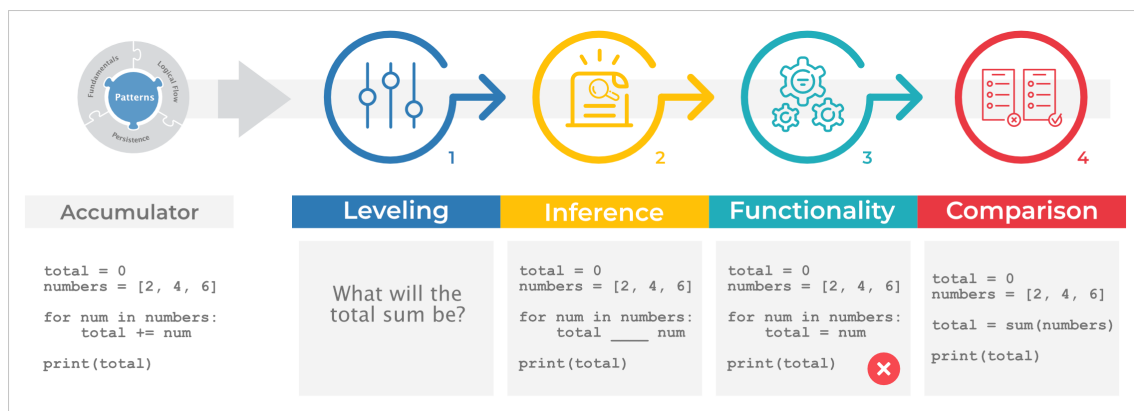


Figure 3. The LIFC Process.

4.1. Interpreting Evidence

In this work, diagnosis represents the process of interpreting the evidence produced by the students, whereas the LOAD profiles represent the results inferred from this process. In this case, the diagnosis is not based on a single instrument, nor is it based on a single correct or incorrect answer. Profiles are identified based on the evidence produced in the different phases of the process. Thus, recurring difficulties associated with certain cognitive competencies make it possible to infer weaknesses related to the dimensions of the 3DC model.

For example, students who demonstrate syntactic mastery but consistently have difficulty predicting the program’s dynamic execution tend to show characteristics associated with the “Literal Textual Comprehension” profile. Likewise, students who are able to perform tracing correctly but have difficulty understanding the functional purpose of the code show evidence associated with the “Operational Comprehension” profile.

In LIFC, the evidence produced by the students is interpreted using diagnostic rubrics associated with cognitive competencies. In this context, rubrics function as support mechanisms for identifying recurring evidence patterns related to the dimensions of the 3DC model. Thus, the same activity can produce syntactic, operational, and functional evidence at the same time.

For example, in an activity involving the accumulator pattern, a student may correctly follow the update of the variable “total” during program execution, demonstrating evidence related to the “Control-Flow” dimension. However, the student may have difficulty perceiving that the code is calculating, for example, the total sum of values in a list, indicating weaknesses associated with the “Functional” dimension.

In this way, rubrics help transform observable evidence into inferences used in the construction of diagnostic profiles. Rather than representing only performance, the process seeks to interpret how different cognitive competencies interact during programming activities.

5. Experimental Methodology

Before conducting the experiment with students, an exploratory evaluation was performed with expert professors, with the aim of analyzing the conceptual clarity, diagnostic coherence, and pedagogical feasibility of the proposal. Three professors with more than 15 years of experience in the introductory programming education participated in this evaluation. They assessed the 3DC model, the LOAD profiles and the LIFC process as coherent, understandable, and potentially useful to support the diagnosis of code comprehension.

The experiment with students was organized into four sequential stages, following the structure of the LIFC process. Each stage was designed to produce evidence related to the Syntactic, Control-Flow, and Functional dimensions of 3DC, as follows:

1. **Stage 1 (Leveling):** Students answered objective multiple-choice questions related to basic programming structures. The objective of this stage was to identify initial evidence related to syntactic interpretation, code execution, and functional understanding of small program snippets.
2. **Stage 2 (Inference):** In the second stage, students performed fill-in-the-gap activities in incomplete code snippets. The objective of this stage was to analyze students' ability to infer missing code segments from the context of the code, producing evidence related to the Syntactic and Control-Flow dimensions.
3. **Stage 3 (Functionality):** In the third stage, the students received the statement of a computational problem and examples of input and output. Initially, participants were asked to produce new test cases for the given problem. Then, they received a reference code and had to analyze whether it would correctly solve the proposed problem. This reference code could be semantically correct or not; that is, it contained no syntactic error. If students considered the code correct, they performed a guided maintenance activity. If they considered the code incorrect, they were asked to carry out an activity involving the correction of logical errors.
4. **Stage 4 (Comparison):** In the fourth and final stage, the students participated in a functional equivalence analysis activity. In this activity, they received the correct code for the proposed problem and had to compare it with different implementations related to the same problem. The different implementations were based on superficial changes, redundancies, and logical errors. The objective of this stage was to investigate evidence related to algorithmic abstraction, functional equivalence, and knowledge transfer.

A total of 35 novice programming students participated in the experiment. All participants were enrolled in an introductory Python-based Computer Science (CS1) course and had already studied fundamental programming concepts and control flow structures.

The study was organized into four stages. Stage 1 consisted of 12 multiple-choice questions (67 patterns), while Stage 2 used four code snippets containing fill-in-the-gap activities (25 patterns). Stage 3 was divided into three sub-stages, each containing two programming problems, totaling 14, 24, and 12 patterns, respectively. Finally, Stage 4 involved functional equivalence analysis activities using five code implementations for each problem, generating 46 patterns. This distribution allowed the same concepts to be observed from different perspectives throughout the LIFC process.

The LOAD profiles were assessed based on the recurrence of evidence produced by the students throughout the different stages of the experiment, considering the Syntactic, Control-Flow, and Functional dimensions of the 3DC model. To support the application of the experiment, a computational tool was developed to collect the evidence produced during the tasks in each stage. This tool is intended exclusively for research purposes and is not yet available to the public. It will be released in a future version after undergoing further refinement and validation.

The experiment was conducted in a controlled environment, and the students did not have access to the Internet or any other type of devices. As a form of motivation, the result was factored into the course assessment.

6. Results

Before analyzing the results, the responses produced by the students in the different stages of the LIFC process were transformed into observable evidence of code comprehension. To this end, each activity in the experiment was previously analyzed with the aim of identifying the “Computational Patterns” required for its completion. The selection of patterns was considered the predominant cognitive objective of each stage, so only elements directly related to task comprehension were used in the analysis.

Each identified pattern was associated with one of the dimensions of the 3DC model: Syntactic, Control-Flow, or Functional. The students’ responses were then analyzed to verify whether they presented evidence of mastery or difficulty in the evaluated patterns. Thus, the unit of analysis used in this study was not the individual question but the evidence produced about each pattern throughout the different stages of the experiment. This strategy made it possible to aggregate evidence from different tasks within the same model dimension, enabling a more comprehensive analysis of the code comprehension of the participants.

Based on the evidence collected, it was possible to calculate the proportion of correct and incorrect answers produced by the students in each dimension of the 3DC model throughout the different stages of the LIFC process. Since each stage was designed to observe distinct manifestations of code comprehension, the analysis of the percentages of correct answers makes it possible to investigate how participants’ performance varies as the tasks require increasing levels of interpretation, validation, diagnosis, maintenance, and abstraction.

Fig. 4 presents the individual performance of the students in the three dimensions of the 3DC model. Overall, the Syntactic dimension showed the highest accuracy rates, with high values for most of the participants. This result suggests that students find it easier to recognize and interpret syntactic elements of the language.

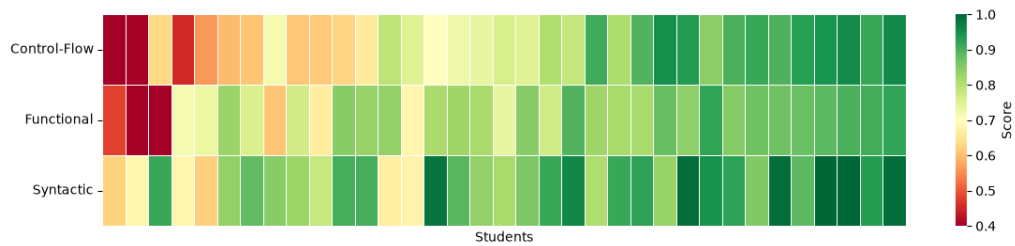


Figure 4. Student Performance in the 3DC Dimensions

On the other hand, the Control-Flow and Functional dimensions showed greater variability among students. This indicates that some participants encountered difficulties in understanding the dynamic execution of programs and, especially, in relating code behavior to the problem it seeks to solve.

It is also possible to observe that while some students showed high and balanced performance across the three dimensions, others obtained good results only in the syntactic dimension. This behavior reinforces the multidimensional nature of code comprehension and suggests the existence of different comprehension profiles among the participants, as proposed by the 3DC model. These findings also suggest that comprehension skills evolve at different rates, highlighting the importance of assessing each dimension independently rather than relying on a single global measure.

To complement the visual analysis presented in the heatmap, Figure 5 presents the distribution of the students’ performance in each dimension of the 3DC model through boxplots. Descriptive statistics are presented in Table 1. Although the heatmap allows one to observe the individual behavior of participants, the boxplot provides an aggregated view of the distribution of results, highlighting measures of central tendency and dispersion.

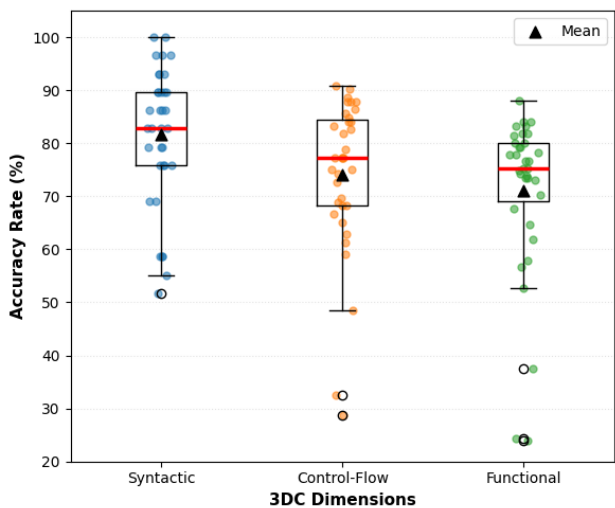


Figure 5. Distribution of Student Performance Across 3DC Dimensions

Statistic	Syn.	CF	Func.
Mean	81.58	74.03	70.98
Q1	75.86	68.20	69.03
Median	82.76	77.27	75.22
Q3	89.66	84.47	80.09
Min	51.72	28.79	23.89
Max	100.00	90.91	88.05
SD	13.06	14.75	15.55

Table 1. Descriptive statistics across the 3DC dimensions.

It can be observed that the Syntactic dimension showed the highest median and mean performance among the three dimensions, indicating that, in general, students demonstrated greater mastery of competencies related to the interpretation of code structure.

In contrast, the Control-Flow and Functional dimensions showed lower means and medians, as well as a greater dispersion in the results. The presence of extreme values in these dimensions indicates that some students had significant difficulties in understanding the dynamic execution of programs and relating their behavior to the functional objective of the problem.

These results corroborate the evidence observed in the heatmap and reinforce that code comprehension is not distributed homogeneously across the dimensions of the 3DC model, revealing different levels of proficiency among the participants.

After the aggregated analysis of the dimensions of the 3DC model, an individual analysis was carried out with the aim of illustrating how the evidence produced throughout the LIFC process can be used to infer LOAD profiles. Unlike previous analyzes, which present general class trends, this evaluation seeks to demonstrate how the recurrence and stability of evidence make it possible to build more detailed diagnoses of a specific student's code comprehension. The following example illustrates how the proposed taxonomy was applied to categorize the students' responses. Due to space limitations, not all responses are presented.

The student analyzed showed a performance of 86.2% in Syntactic, 48.5% in Control-Flow, and 24.3% in Functional, revealing a significant difference among the three dimensions. Although the student demonstrated strong mastery of competencies related to the interpretation of the code structure, the performance in competencies associated with understanding the execution of the program and the functional purpose was substantially lower.

The analysis of the LIFC stages showed that several correct answers obtained in the initial activities were not confirmed in later stages, especially when tasks started to require inference, diagnosis, maintenance, and functional comparison. In addition, persistent difficulty and instability were observed in several patterns associated with the Control-Flow and Functional dimensions, whereas the syntactic patterns remained consistently confirmed.

Overall, these pieces of evidence suggest characteristics compatible with the Operational (O) profile, in which the student is able to adequately recognize syntactic elements of the code, but has difficulty constructing a consistent mental model of its execution and relating this behavior to the computational problem that the program seeks to solve.

Using the minimum-dimension criterion, most students were classified as L (46%) or O (40%), while 14% were classified as A and none as D. Since the LOAD profile is defined solely by the student's lowest-scoring dimension, these results depend directly on the adopted classification strategy. Investigating alternative classification methods is left for future work.

7. Conclusions

This work presented the Three-Dimensional Comprehension (3DC) model, a multidimensional proposal for representing and diagnosing code comprehension among novice programming students. Unlike approaches that assess this ability through isolated tasks or solely based on final performance, the model organizes the evidence produced by students into three complementary dimensions: Syntactic, Control-Flow, and Functional.

To operationalize the model, the LIFC process was proposed, integrating different assessment instruments into a structured sequence of code reading, inference, functional analysis, and code comparison activities. Based on this evidence, the LOAD diagnostic profiles were defined, allowing interpreting recurring patterns of comprehension and difficulty presented by the students.

The results obtained indicate that code comprehension does not manifest itself homogeneously across the evaluated dimensions. Overall, students performed better in the Syntactic dimension, while more significant difficulties were observed in the Control-Flow and Functional dimensions. In addition, individual analysis showed that the evidence produced at different stages of the process can reveal difficulty patterns that would not be identified by traditional assessments based only on correct answers or final scores.

In response to RQ1, the results suggest that code comprehension can be represented through a multidimensional model capable of characterizing different learning profiles. Regarding RQ2, the results indicate that the integration of multiple instruments and evidence allows more detailed and interpretable diagnoses of students' difficulties to be constructed.

As its main contribution, this work proposes a structured way to transform observable evidence produced during programming activities into more informative pedagogical diagnoses. The 3DC model, the LIFC process, and the LOAD profiles are expected to support teachers and researchers in identifying specific code comprehension difficulties and in planning more targeted pedagogical interventions.

In future work, we intend to expand the experimental application with a larger number of participants and continue investigating the stability of the LOAD profiles. We also intend to correlate the results with development activities. Finally, we will recommend pedagogical interventions for each LOAD profile.

8. Threats to Validity

This study presents some limitations. The main threat is related to the interpretation of the evidence used to represent the dimensions of the 3DC model and infer the LOAD profiles, since this process involves decisions based on rubrics and on the mapping of Computational Patterns. In addition, factors such as motivation, attention, and interpretation of the problem statements may have influenced some of the students' responses.

9. Statement on the Use of Artificial Intelligence

During the preparation of this article, the ChatGPT tool was used to support grammatical review, reorganization of textual passages, and suggestions to improve the writing of the article. All decisions related to study design, data analysis, interpretation of results, and conclusions were made by the authors.

References

Andersen-Kiel, N. and Linos, P. P. (2024). Using ChatGPT in Undergraduate Computer Science and Software Engineering Courses: A Students' Perspective. In *2024 IEEE Frontiers in Education Conference (FIE)*, pages 1–9.

- Banweer, K. and Trytten, D. A. (2024). WIP: Code Insight: Combining Code Reading and Debugging Practices for Active Learning in Entry-Level Computer Science Courses. In *2024 IEEE Frontiers in Education Conference (FIE)*, pages 1–5.
- Cavalcanti, V. and Andrade, W. L. (2025). Code Comprehension for Novice Students: Teaching, Assessment, Tools, and Challenges. In *2025 IEEE Frontiers in Education Conference (FIE)*, pages 1–9.
- Feitelson, D. G. (2021). Considerations and Pitfalls in Controlled Experiments on Code Comprehension. In *Proceedings of the 29th IEEE/ACM International Conference on Program Comprehension (ICPC)*, pages 106–117. IEEE.
- Hannebauer, C., Hesenius, M., and Gruhn, V. (2018). [Journal First] Does Syntax Highlighting Help Programming Novices? In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 704–704.
- Izu, C. and Mirolo, C. (2020). Comparing Small Programs for Equivalence: A Code Comprehension Task for Novice Programmers. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '20*, page 466–472, New York, NY, USA. Association for Computing Machinery.
- Kallia, M. (2023). The Search for Meaning: Inferential Strategic Reading Comprehension in Programming. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1, ICER '23*, page 1–14, New York, NY, USA. Association for Computing Machinery.
- Lehtinen, T., Seppälä, O., and Korhonen, A. (2023). Automated Questions About Learners' Own Code Help to Detect Fragile Prerequisite Knowledge. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1, ITiCSE 2023*, page 505–511, New York, NY, USA. Association for Computing Machinery.
- Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., and Thomas, L. (2004). A multinational study of reading and tracing skills in novice programmers. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education, ITiCSE-WGR '04*, page 119–150, New York, NY, USA. Association for Computing Machinery.
- Pennington, N. (1987). Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, 19(3):295–341.
- Schulte, C. (2008). Block Model: an educational model of program comprehension as a tool for a scholarly approach to teaching. In *Proceedings of the Fourth International Workshop on Computing Education Research, ICER '08*, page 149–160, New York, NY, USA. Association for Computing Machinery.
- Shneiderman, B. and Mayer, R. (1979). Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International Journal of Computer; Information Sciences*, 8(3):219–238.
- Szabo, C. (2015). Novice Code Understanding Strategies during a Software Maintenance Assignment. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 276–284.

- Teague, D. and Lister, R. (2014). Programming: reading, writing and reversing. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, ITiCSE '14, page 285–290, New York, NY, USA. Association for Computing Machinery.
- Wilson, D., Sarwar, S., and Najjar, N. (2024). Notional Machines for Inclusive Learning. In *2024 IEEE Frontiers in Education Conference (FIE)*, pages 1–9.
- Wyrich, M., Bogner, J., and Wagner, S. (2023). 40 Years of Designing Code Comprehension Experiments: A Systematic Mapping Study. *ACM Comput. Surv.*, 56(4).
- Zavala, L. and Mendoza, B. (2017). Precursor skills to writing code. *J. Comput. Sci. Coll.*, 32(3):149–156.