Navigating Software at Different Abstraction Levels

Pablo Sabater-García[1,2,*,†], Gregorio Robles-Martínez[3,†]

[1]Universidad Rey Juan Carlos, Madrid, Spain

Abstract

Program comprehension is the process and result of building an accurate mental model of software. Nowadays, AI-assisted programming increases the challenge of understanding code, because of not being authored by the person who seeks to understand, and the increasing volume of code to be understood. These challenges and difficulties motivate the need for using tools for assisting the task of program comprehension, by means of visual representations and verbal explanations, which currently work independently. We have observed a gap between the high-level given by the visual representations and the underlying code. Our contribution is an approach which bridges this gap, by means of LLM-generated explanations and visualization of the dynamic behavior of the application. We have created a prototype with the Scratch language and the *CodeCity* metaphor, providing brief and context-aware LLM-explanations of the code structures represented as districts, neighborhoods, buildings, and relationships between parts of the program, and visualizing the broadcast and variable traffic. This makes it possible to navigate across different abstraction levels. This approach aims to reduce the cognitive load involved in understanding, maintenance, and evolving software.

Keywords

Software Visualization, CodeCity, Program Comprehension, Code Comprehension, Program Understanding Tools, Abstraction Levels

1. Motivation

Program comprehension is the process and result of understanding the structure and behavior of a software application [1]. This process is necessary for software maintenance and evolution. It is considered that a developer understands a program when they are capable of connecting code structures to their real-world knowledge, and explain the program's behavior, effects, and domain relationships [2]. The study of program comprehension relies on different theoretical approaches, such as the mental and the cognitive model, and strategies of comprehension are divided into top-down and bottom-up, which developers strategically combine [3].

Program comprehension support tools may be divided into graphical visualizations, such as *CodeCity* [4], and textual explanations or descriptions, modalities which are usually developed independently. Visualizations offer a high-level overview, but lack detail, and the opposite occurs in the textual modality. Separation of both modalities creates an abstraction gap between the high-level visualizations and local knowledge of a program [5]. To bridge this gap, we propose an approach that combine structural visualization of *CodeCity* with context-aware descriptions and dynamic dependency highlighting, and therefore reduces the cognitive effort of understanding, maintaining and evolving software.

2. Proposed approach

Our approach consists of bridging the gap between the high-level visualization given by the city metaphor and the source. The approach includes context-aware descriptions in diverse granularity levels, each one having a maximum extension of 100 words, in order to avoid a cognitive overload, and with both a verbose and a schematic version. Describable elements are all things that are represented in the *CodeCity* metaphor, such as districts, neighborhoods, and buildings, and every element which

*Corresponding author.

†These authors contributed equally.

pablo.sabater@urjc.es (P. Sabater-García); grex@gsyc.urjc.es (G. Robles-Martínez)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

creates a relationship between parts of the application. Every time that in a description it is mentioned another describable element, a hyperlink is created to its description. In addition, communication events and shared state variables are highlighted via building coloring. The goal that our approach achieves is a cross-modal navigation between the panoramic city view, the intermediate descriptions, and the underlying source code. In the next section we present a prototype in scratch to evaluate our approach. However, the approach is applicable to other languages.

3. Prototype

We have implemented an ad hoc prototype for a scratch program for testing our approach. Given the block-based nature of scratch, low-level syntax is hidden and the user is capable of focusing on the higher-level logic avoiding the obstacle of paying attention to complex syntax. We have used the visualization criteria given in [6], by adding descriptions and dynamic visualizations. In this metaphor districts are groups of sprites sharing some functional features, neighborhoods are the sprites and buildings are the scripts. The user may navigate across the city by mouse and WASD keys. It includes context-aware LLM-based descriptions. Descriptions have a verbose and a schematic version, each in its own tab in the description window, and they do not exceed 100 words in order to avoid cognitive overload. There is included a description of the whole program, a description for each district, each neighborhood and each building, in addition, variables, broadcasts, and lists have their own description. Every time that in a description another describable element is mentioned, a hyperlink allows the user to navigate to its description. In addition, the prototype includes a panel with all broadcasts, variables, and lists, and when one of them is selected, buildings implied in the respective relationships are highlighted by colors. When nothing is selected in the panel, the highlighting indicates dead code.

4. Conclusion

The proposed approach bridges the abstraction gap between *CodeCity* and source code, by enriching the visualizations with descriptions and dynamic dependency highlighting. This reduces the cognitive load associated with reviewing foreign or AI-generated code, and therefore facilitates software maintenance and evolution. Our immediate next step is a controlled proof of concept in order to validate the expected improvement. Another future goal is to explore the adaptation of our approach to other languages, such as Python or C, adapting the visual metaphor to represent function calls and module imports.

References

[1] W. Maalej, R. Tiarks, T. Roehm, R. Koschke, On the comprehension of program comprehension, ACM Transactions on Software Engineering and Methodology (TOSEM) 23 (2014) 1–37.

[2] T. J. Biggerstaff, B. G. Mitbander, D. Webster, The concept assignment problem in program understanding, in: Proceedings of the 15th International Conference on Software Engineering, ICSE '93, IEEE Computer Society Press, Baltimore, Maryland, USA, 1993, pp. 482–498. doi:10.1109/ICSE.1993.346017.

[3] M.-A. Storey, Theories, methods and tools in program comprehension: past, present and future, in: 13th International Workshop on Program Comprehension (IWPC'05), IEEE, 2005, pp. 181–191.

[4] R. Wettel, M. Lanza, Visualizing software systems as cities, in: 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, IEEE, 2007, pp. 92–99.

[5] H. M. Kienle, H. A. Muller, Requirements of software visualization tools: A literature survey, in: 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007, pp. 2–9. doi:10.1109/VISSOF.2007.4290693.

[6] G. Robles, D. Escobar-Morales, S. Montes-León, J. Moreno-León, G. Pons-Castro, D. Moreno-Lumbreras, Towards visualizing educational block-based scratch code with the city metaphor, in: 2025 IEEE Working Conference on Software Visualization (VISSOFT), IEEE, 2025, pp. 58–63.