International Journal on Advanced Science Engineering Information Technology

Vol.13 (2023) No. 4
ISSN: 2088-5334

Multi-Language Program Understanding Tool

Navid Rostami Ravari [a], Rodziah Latih [a,*], Abdullah Mohd Zin [b]

[a] Center for Software Technology and Management, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, 43600 UKM Bangi, Malaysia
[b] Faculty of Computing and Multimedia, Universiti Poly-Tech Malaysia, Kuala Lumpur, Malaysia
Corresponding author: *rodziah.latih@ukm.edu.my

Abstract—**Open-source programs have gained popularity due to their decentralized, quick development cycles and accessibility to everyone. Program understanding is vital for open-source software developers to modify or improve the code. However, one problem open-source developers face is the difficulty in understanding the programs as the program grows large and becomes complex. The current program understanding tool is inefficient because it only supports one programming language, while open-source programs are written in various languages. This paper discusses a new program understanding technique that facilitates multi-language program understanding. The proposed technique helps developers to understand open-source programs by supporting two unique features: multimedia and additional comments. We carried out this study in four stages. First, we examined available tools and techniques in software understanding to identify their strengths and weaknesses. Second, we proposed a new technique. Third, we designed a new tool to implement the new technique. Lastly, we evaluated the tool using a survey. We invited twenty users, including students and programmers, to use the system and ask for their feedback. The evaluation of the proposed techniques shows that the respondents have a positive perception as they agree that the technique helped them better understand the program. The multimedia support and an additional comment provided by the tool significantly improve user understanding of the program. For future work, we would like to explore the possibility of utilizing some machine-learning techniques to enhance the process of program understanding.**

Keywords— **Program comprehension; program visualization; open-source software; source code; multimedia.**

Manuscript received 8 Jun. 2022; revised 12 Feb. 2023; accepted 16 Mar. 2023. Date of publication 31 Aug. 2023.

I. Introduction

Open-source software (OSS) is developed to be decentralized and collaborative, relying on peer review and community production. The software development company contributed to OSS projects for many reasons, including improving software quality and a desire to influence the software's development direction [1]. The most commonly mentioned benefit of open-source software is cost saving. Typically, open-source software implies that users are not obligated to pay for software and procurement overhead to manage license renewals [2].

Open-source software is usually easier to get than proprietary software. It is more reliable because thousands of independent programmers voluntarily test and fix software bugs [3]. Open source is independent of the company or author that created it. Even if the company fails, the program continues to exist and be used. Also, open-source software uses open standards that are accessible to everyone. Thus, it does not have the incompatible formats problem that may exist in proprietary software. However, lack of support, poor documentation, and program complexity cause severe problems when using open-source software [4], [5]. The high volume of information and missed and conflicting information makes understanding the software programs difficult. As a result, not many people can utilize open-source software.

Program understanding is vital for open-source software developers who want to modify or improve the code. Open-source software is becoming large and complex because its development and maintenance involve the collaboration of many people. Therefore, understanding, modifying, and expanding the open-source software become more complex and require significant time. In particular, programs that have evolved over many years are challenging to understand because diverse programmers with different programming styles have maintained them and have evolved to become unnecessarily complex and extensive.

The majority of open-source programs comprise a high volume of code in a large number of files. Therefore, inadequate documentation makes understanding and using the source code impossible or very difficult. A reasonable solution is a program understanding tool that explains the purpose and task of different parts of the source code.

Comprehending large-scale software costs more and takes more effort [6]. The academic literature mainly focuses on research in program comprehension of short code, but the comprehension of large-scale software is more critical and needs to be considered. The open-source software may differ in system comprehension, and further research is required to understand it. This study is essential because open-source development is more evolutionary, less planned, and less documented than large-scale software [7].

This paper discusses a new technique to facilitate open-source software understanding. The paper is structured as follows. Section 2 discusses the related work in program understanding and the method employed for the study. Section 3 reports the results and discusses the finding. Finally, Section 4 presents our conclusions and suggestions for further research.

II. Materials and Method

A. Program Understanding Techniques

Program understanding is essential for software maintenance and enhancement activities [8], [9]. It involves comprehending programs to perform further tasks such as fixing bugs, refactoring code, and porting code to different platforms. Generally, program understanding comprises three steps: reading the documents and source code and running the source code [10]. For the first step, documentation is very useful for studying the behavior of an executable program. In this case, a lack of documentation or poor documentation prevents developers from understanding the systems effectively. Therefore, they must go through the second and third steps, i.e., read and run the source code to understand how a program was designed and worked.

Various techniques and tools have been developed to facilitate the program understanding strategies programmers use to reach specific tasks. Five strategies for program understanding are:

1) The "bottom-up" or "chunking": This strategy involves comprehending the program's "bottom-up" by reading the source code and then mentally dividing low-level software artifacts into meaningful, higher-level abstractions [11]. This semantical group is called chunks. These secession processes are continuously done until the program becomes highly understood. This strategy is suitable when programmers know only the insufficient program domain.

2) The "top-down": This theory proposes that programmers use their experience and repeatedly try to certify their expectations based on their design [12]. The programs are comprehended "top-down" by rebuilding knowledge about the scope of the application and mapping it to the source code. For example, the programmer decomposes the new operating system into familiar elements, like a file manager, process manager, I/O manager, and memory manager. This strategy is suitable when the program or type of program is familiar.

3) Knowledge-based: This strategy is known as the Letovsky Model [13]. It possesses three components: a knowledge base, a mental model, and an assimilation process. The knowledge base consists of programming expertise, problem-domain knowledge, rules of discourse, plans, and goals. The first component encodes the expertise and knowledge the programmer brings to understand the task. A programmer's primary understanding of the target program is encoded in the second component. It should be noted that the mental model changes during the understanding process. Finally, through an assimilation process, the knowledge base is associated with the target program code and documentation to improve the mental model.

4) Integrated approaches: This approach merges the top-down structure model, situation model, program model, and knowledge-based method into a meta-model [14]. The top-down, situation, and program models reflect the comprehension process. In contrast, the knowledge-based model furnishes the process with information related to the comprehension task and stores any new and inferred knowledge. Some programmers regularly switch between these three models. By moving freely between these three strategies, understanding the program code is built simultaneously on several levels of abstraction.

5) Task-based approach: This approach is suitable for novice developers to enhance program comprehension [15]. Tasks can be defined in the lower cognitive category, such as recall, or the higher cognitive category, e.g., source code modification.

B. Program Visualization

Visualization transforms information into a visual form, enabling users to observe the information [16], [17]. Generally, visualization can be categorized into three groups: (i) structure, which includes visualizations that support the analysis of the static aspects and relationships in software systems, (ii) behavior, which relates to visualizations proposed for the data analysis collected from the execution of programs, and (iii) evolution, which contains visualizations that support the analysis of how systems change over time [18]. A city metaphor and a directed acyclic graph can represent the structure of software systems. In contrast, the system's behavior is visualized using log traces, and the system's evolution is visualized with code change history [19].

Program visualization is one of the programs understanding approaches. Program visualization tools make the program visible by displaying the structure and elements of the source code. The illustration can help users remember and identify how the code works. As a result, programmers can better understand or remember their code. For novice programmers, it can remove the barriers to discovering how the code and algorithms work [20].

C. Program Understanding Tools

Merely reviewing the code does not facilitate an understanding of large programs. Much information could be easily lost, including object-oriented inheritance hierarchies, particular employment of class approaches, and the attendance or non-attendance of specific design patterns. To facilitate this process, software visualization tools were suggested. Hunter is a visualization tool for JavaScript applications [21]. It visualizes source code through a set of coordinated views that include a node-link diagram that depicts the dependencies among the components of a system and a tree map that helps programmers to orientate when navigating its structure.

Program visualization tools can be used to support analysis, modeling, testing, debugging, and maintenance activities [22]. Simple and helpful visualization tools can significantly reduce the effort spent on program understanding and maintenance. On the other hand, inappropriate and inefficient software visualization tools create complexity that prevents proper understanding of the program code. These inappropriate tools lead to confusion and misunderstandings for users. Furthermore, most available program comprehension tools focus primarily on showing graphical elements of source code rather than facilitating source code comprehension. Although many program understanding tools have been developed, most tools only support specific programming languages. Different programming languages have different complexities. The difficulty of importing and exporting source code is also a usability issue for these tools.

Software understanding tools are practical if the tools are easy to use and help users to achieve results faster than the traditional approach. Strong code understanding support can simplify tasks like improving documentation, maintenance, testing, adding new functionalities, debugging, and analyzing code. Besides using graphical presentation, program visualization can also be realized using techniques such as node-link diagrams, graphs, infographics, and tree-map [17], [20], [22], [23].

However, software visualization still has several issues, e.g., software scalability, tools validation technique, and scope-related vision [24]. Most software understanding tools were developed as short-term research prototypes or "toy programs" and do not fit the industrial scope in terms of the program inputs range. In this regard, the expectation is that the software understanding tools are only suitable for small- to medium-sized systems. We should focus on a production scale system to potentially deploy software understanding in the industry.

The lack of rigorous validation techniques is one of the main problems discussed in the research of software understanding tools. Beyond the idea of effectiveness, most research did not articulate research methods and questions. Indeed, surveys or controlled experiments are not a popular evaluation approach compared to case studies, even though it is a proper validation process.

Experts in reverse engineering, reengineering, and software preservation believe software visualization, especially 3D visualization, is too metaphorical. The researcher should understand that software visualization aims not to create impressive images but to use images to evoke viewers' mental images for better understanding. The software understanding system should represent more knowledge of the application area to envision the software in context.

Software visualization also lacks usability. To solve this issue, a researcher should consider human factors when designing and evaluating software understanding tools mainly employed by teachers in the education domain. These tools are supposed to encourage active learning for students. Hence, conducting empirical studies on the current understanding and considering such validation in designing future understanding systems is essential.

In conclusion, from the literature review and some comparisons of software understanding techniques, it is clear that program understanding tools can assist users in understanding the source code better. In addition to the benefits, we identified various issues that made users reluctant to use the tools. We use these issues as guidelines for developing new program comprehension techniques.

D. Research Method

We conducted this study in four stages. First, we review available program understanding tools and techniques to identify strengths and weaknesses. Second, we propose new techniques based on the research results obtained. Third, we design and develop the program comprehension tool using the .Net framework with C# programming language and MySQL database. Finally, we evaluated the developed program comprehension tool using a survey approach. We invited a group of students and programmers to use the system and give their feedback.

III. Results and Discussion

This section discusses the literature study's results, new program understanding technique, tool, and result evaluation.

A. Program Understanding Technique

We propose a new program comprehension strategy based on a literature review and analysis of several programming comprehension tools. This strategy aims to help programmers better understand open-source software. Understanding open-source software is challenging as it grows in size and complexity due to the involvement of many people. Thus, the proposed program comprehension technique has four features that consider this challenge and are explained below.

1) Support Multiple Programming Languages: Most program understanding tools depend on a particular programming language. Therefore, they can only visualize program execution in that language [25], [26]. Unfortunately, one language cannot meet all the requirements of the software industry. At least one-third of the current software programs employ two programming languages, and 10% of all applications include three or more languages [27].

Large software systems are usually programmed in several languages. For example, the core of powerful software is written in languages such as Java, C, or Python, while the user interface is written in languages such as JavaScript, Python, Perl, or other scripting languages. The reason for using different languages in one application is that each language has its strengths.

This trend has an impact on software understanding tools. A multi-language tool that supports a combination of programming languages can enhance the validity of the proposed tool because source code is usually written in different languages. Moreover, a multi-language software understanding tool will increase user satisfaction.

2) Understand the Program Structure: Understanding a big and complex program without understanding the program's structure is challenging. Thus, a program understanding tool must provide this facility. Types of information that must be provided include each identifier in the program, the relationship between identifiers, and subprograms, and their relationship with other subprograms.

3) Source code tagging using multimedia: Professional and novice programmers must check the source code to comprehend the program. Depending on the individual's expertise and the program's complexity, this process may take time. Usually, the best way to comprehend the program is to use updated documents or physically communicate with expert developers. However, documents may be outdated and do not include new changes to the program or may not provide access to expert developers. Maintaining the experts' knowledge or information about the program is necessary by allowing the community of programmers to tag information to various parts of the source code. Other programmers can then use this information to understand it.

One of the effective tagging methods is visual media, like written media and drawing, audio media, audio-visual media (e.g., video and animation), and multi-sensory media (e.g., 3D objects and simulations). The media that the expert developer added allow other programmers to comprehend program changes. Additionally, it helps explain why they use specific programming techniques in the various sections. For example, in large companies where different people work on the same source code, each person can add media to their work orally (audio) or by explaining the source code through video [28]. Therefore, this technique assists the new programmer in comprehending the program in less time and at a lower cost.

4) Additional comments to Enhance understanding: Another meaningful way to program understanding is by writing comments in the source code [29]. Comments are generally formatted as either block comments or line comments (also called inline comments). Comments usually provide additional algorithm information, specify constraints, or warn developers about code complexity [30]. Without proper comments, it is not easy to understand the source code [31]. However, using comments on source code is often overlooked, even though developers know the benefits.

Sometimes, developers forget to update the relevant comments when changing a part of the program or function. Thus, the comments might adversely affect the success of software evolution and the process of program understanding. Comments like this often mislead developers and create bugs in the future. Also, open-source code with a high-density comment is more likely to cause problems understanding the source code. Comment density is the percentage of comment lines in the source codebase, or in other words, comment lines divided by total code lines. The appropriate size of comment density is likely to be an element of software survival. However, when comments become large, they complicate the perception of the source code, resulting in the opposite. Additional comments are an additional tagging feature to the source code. Additional comments allow other programmers to explain various parts of the source code, for example, why certain design patterns are used. Using additional comments can avoid massive inline comments and lengthy block comments.

B. Multi-language Program Understanding Tool

As discussed above, we developed a new program understanding tool based on the proposed technique. The proposed technique includes three supporting elements: multi-language, multimedia, and additional comments. Multi-language is necessary as most programmers use several programming languages, and most open-source programs are written in various languages. Using multimedia, such as PowerPoint, audio, Video, Image, and PDF file format, as a tagging method gives the user additional support to improve program comprehension.

Programmers usually add readable and reliable explanations about complex parts of the program in the form of comments to boost program comprehension. However, many comments among the source code lines are more likely to cause confusion and increase the complexity of the source code. Additional comments can be a solution to this complexity. This tool has two types of users: authors and users. An author is a person who can upload a new source code into the tool and can add supplementary information to the source code. The supplementary information can be video, audio file, PowerPoints, Comment, photo, and PDF, to help the user better understand the source code. A user is a person who uses the tool to understand the source code better.

We developed this tool using the C# programming language and the .NET framework. The tool primarily aims to help users understand and learn the source code faster. Therefore, the essential criteria in designing this user interface are simplicity and ease of use. Fig. 1 shows the flowchart describing how to use this tool. The source code is first imported into the tool (it should be noted that this tool is not a debugging tool, so we assume that the source code is free of syntax errors). After importing the source code, the elements in the source code are extracted using Ctags.

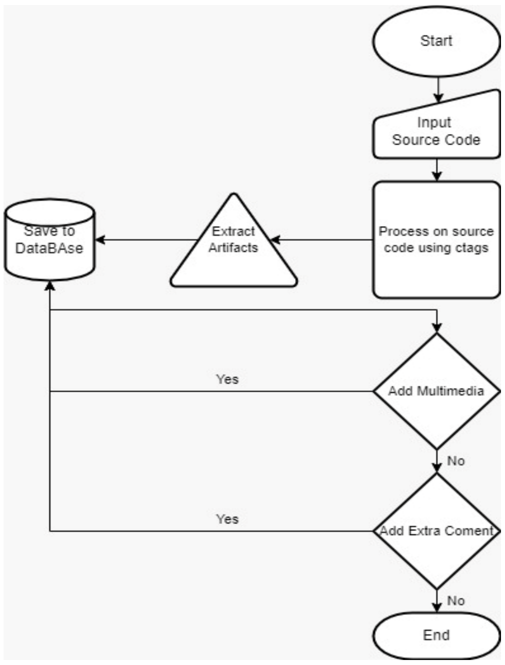
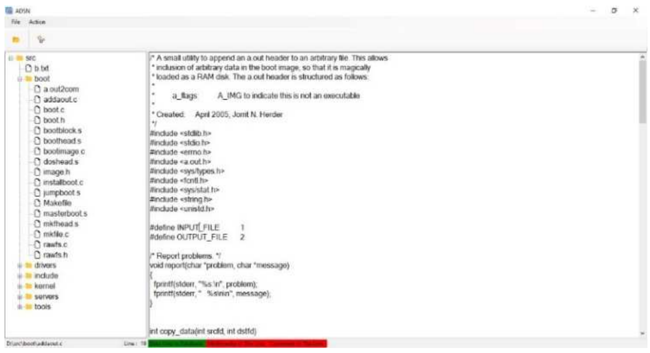


Fig. 1 Flowchart describing the process flow

Ctags is a tool to identify objects in the source code and keep them in a tag file. It supports 41 programming languages, including C, C++, C#, and Python. Ctags generates a cross-reference file that lists the information about the various language objects in a source file. The information extracted by Ctags is stored in the database and can be retrieved in a query format. After this step, the author can add information to the source code according to the line number. The information is either multimedia or comments. Fig. 2 shows the tool's user interface, displaying the uploaded source code.



```
/* A small utility to append an a.out header to an arbitrary file. This allows
 * inclusion of arbitrary data in the boot image, so that it is magically
 * loaded as a RAM disk. The a.out header is structured as follows:
 *
 *      a_flags:      A_IMG to indicate this is not an executable
 *
 * Created:    April 2005, Jorrit N. Herder
 */
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <a.out.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>
#include <unistd.h>

#define INPUT_FILE      1
#define OUTPUT_FILE     2

/* Report problems. */
void report(char *problem, char *message)
{
  fprintf(stderr, "%s:\n", problem);
  fprintf(stderr, "   %s\n\n", message);
}


int copy_data(int srcfd, int dstfd)
```

Fig. 2 User Interface of the software tool

There are two categories of information: multimedia and additional comments. A user right-clicks any line in the source code and chooses the type of information that the user wants to see, as shown in Fig. 3. The status bar at the bottom of the editor shows the availability of supporting information (Fig. 4). A green flag means that the information is available. Otherwise, it is marked with a red flag. If the selected information is available, it is displayed on a separate page. Fig. 5 shows the source code with supporting video and PowerPoint slides. Fig. 6 shows a user interface to add additional comments.

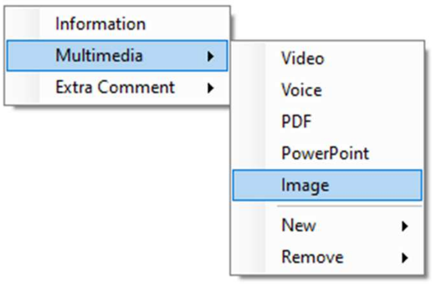


Fig. 3 Showing information in multimedia format

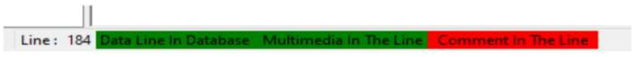
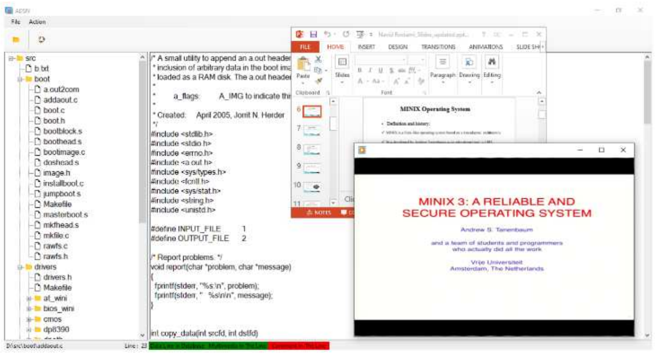


Fig. 4 Different flags relate to three categories of supplementary information.



```
/* A small utility to append an a.out header
 * inclusion of arbitrary data in the boot ima
 * loaded as a RAM disk. The a.out header
 *
 *      a_flags:      A_IMG to indicate thi
 *
 * Created:    April 2005, Jorrit N. Herder
 */
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <a.out.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>
#include <unistd.h>

#define INPUT_FILE      1
#define OUTPUT_FILE     2

/* Report problems. */
void report(char *problem, char *message)
{
  fprintf(stderr, "%s:\n", problem);
  fprintf(stderr, "   %s\n\n", message);
}


int copy_data(int srcfd, int dstfd)
```

MINIX Operating System

- Definition and history:

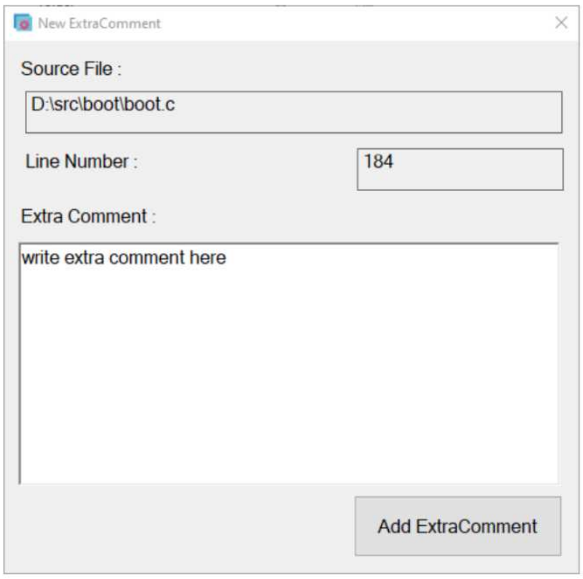
MINIX 3: A RELIABLE AND SECURE OPERATING SYSTEM

Andrew S. Tanenbaum

and a team of students and programmers who actually did all the work

Vrije Universiteit
Amsterdam, The Netherlands

Fig. 5 The source code with supporting video and slide



Source File : D:\src\boot\boot.c

Line Number : 184

Extra Comment : write extra comment here

Fig. 6 Adding new extra comment

C. *Technique Evaluation*

We evaluated the effectiveness of the proposed technique by conducting a survey. We used a five-level Likert scale questionnaire to measure the experts' agreement with the statements. The response scales are 1-Strongly disagree, 2-Disagree, 3-Neither agree nor disagree, 4-Agree, and 5-Strongly agree. The survey involved 20 participants (5 expert programmers and 15 students). A sample size of 20 participants is sufficient because, according to the Usability Test Sample Size Model, most usability problems are detected by the first three to five subjects [32]. Running additional subjects during the same test is unlikely to reveal new information.

There is no age or gender restriction in our survey. The users were introduced to the new program understanding tool and given ample time to try out the tool. Later, we asked them to answer a questionnaire regarding their experience using it. The questionnaire consisted of three parts. The first part is about user perceptions of the proposed technique. This part consists of eight questions: Q1 and Q2 ask the users' perception of source code comprehension, Q3-Q7 ask the users' perception about using multimedia in understanding the source code, and Q8 asks about the users' perception of the proposed technique. Table 1 shows the mean for each question, which is 4.35. The result indicates that the users positively perceive using multimedia in understanding the source code and the proposed technique.

The second part of the questionnaire dealt with how the user interacts with the tool. It consists of five questions concerning ease of use (Q9 - Q10) and user satisfaction with the tool (Q11-Q13). Table 2 shows that the mean value is 4.26, which confirms that users are satisfied with the tool. The third part consists of five questions (Q14-Q18) to evaluate the technique's effectiveness in understanding the source code. The participants were asked to comprehend the MINIX source code using the technique. Table 3 shows the result. The mean for this MINIX case study is 4.0, which indicates that the techniques give a better understanding and learning of the source code.

TABLE I
USERS' PERCEPTION OF THE PROPOSED TECHNIQUE

Questions	Mean
Q1. Has it been difficult for you to understand the source code?	4.15
Q2. Has it become easier for you to understand open-source code using this tool?	4.3
Q3. Do you find it appropriate to use media for learning?	4.9
Q4. Do you find it appropriate to use media to learn source code and programming?	4.6
Q5. Has the use of media in this tool helped you better understand the source code?	4.4
Q6. Do you find it appropriate to use comments outside the source code as additional comments?	4
Q7. Has the additional comment in the tool helped to better understand the source code?	4.3
Q8. Do you see the technique used in this tool as an appropriate way to better understand the source code?	4.3
The average score of questions	4.35

TABLE II
USERS' SATISFACTION WITH THE TOOL

Questions		Mean
Q9.	Has working with tools been easy for you?	4.35
Q10.	Do you believe it can be used without special knowledge about the tool?	4.55
Q11.	Do you want to work on the tool in newer versions?	3.95
Q12.	Do you recommend this tool to your friends?	4.15
Q13.	Was the order of the tool options in the proper order?	4.3
The average score of questions		4.26

TABLE III
THE EFFECTIVENESS OF THE TECHNIQUE

Questions		Mean
Q14.	Do you want to check the source code of the MINIX operating system?	3.05
Q15.	Do you find it difficult to understand the source code of the MINIX operating system?	4.3
Q16.	Did using the tool on the source code of the MINIX operating system lead to a better understanding of it?	4.2
Q17.	Has the technique of adding multimedia helped better to understand the source code of the MINIX operating system?	4.3
Q18.	Has the extra comment technique helped better to understand the source code of the MINIX operating system?	4.15
The average score of questions		4.0

IV. CONCLUSION

Program understanding is one of the most critical tasks in using source code. The recent open-source programs are complex and complicated to understand because they were developed by many programmers using different languages and styles. Techniques that have been developed to understand programs have different strengths and weaknesses. The weaknesses of existing techniques motivate us to introduce a new technique to improve the understanding of open-source software.

Our proposed technique simplifies the understanding of open-source programs by supporting two unique features, i.e., the ability to add multimedia and additional comment to the complex open-source code. These additional features help with a better understanding of the source code. Moreover, the tool we developed also supports multiple programming languages to help users examine source code written in different languages.

The evaluation of the proposed techniques shows that the users have a positive perception because they agree that the technique is better at assisting them to understand the program. They also agree that it is easy to use. The tool's multimedia support and extra comment significantly improve user understanding of the source code. This proposed technique can be used via GitHub and design proper plugins for IDEs, such as Eclipse or IntelliJ IDEA. Users who access the source code from GitHub receive the multimedia and supplementary comment assigned to it.

The proposed software understanding tool currently supports five media types: video, audio, image, PDF, and PowerPoint. This tool considers a wide range of available media and their unique use. Each media can be used to improve source code understanding. In its current form, the software tool suffers several limitations. One of them is the lack of intelligence to some understanding process to be carried out automatically. For future research, we would like to explore the possibility of using some machine learning algorithms that can help enhance the program understanding process.

ACKNOWLEDGMENT

We thank Universiti Kebangsaan Malaysia for supporting this work under the GGPM-2020-026 research grant fund.

REFERENCES

[1] S. Butler *et al.*, "On Company Contributions to Community Open Source Software Projects," *IEEE Trans. Softw. Eng.*, vol. 47, no. 7, 2021.

[2] A. Khandelwal, "Impact of Open Source Software in Research," 2020.

[3] A. Azlen, M. Nordin, R. Latih, and N. M. Ali, "Using SaaS to Enhance Productivity for Software Developers: A Systematic Literature Review," *J. Theor. Appl. Inf. Technol.*, vol. 31, p. 24, 2020.

[4] Sumandeep Kaur, "Issues in Open-Source Software ," *Int. J. Comput. Sci. Commun.*, vol. 11, no. 2, pp. 47–51, 2020.

[5] G. M. Kapitsaki, N. D. Tselikas, K.-I. D. Kyriakou, and M. Papoutsoglou, "Help me with this: A categorization of open source software problems," *Inf. Softw. Technol.*, vol. 152, p. 107034, Dec. 2022.

[6] A. Mohd Zin, S. Ahmad Aljunid, Z. Shukur, and M. Jan Nordin, "A Knowledge-based Automated Debugger in Learning System," 2000.

[7] O. Levy and D. G. Feitelson, "Understanding large-scale software systems – structure and flows," *Empir. Softw. Eng.*, vol. 26, no. 3, p. 48, May 2021.

[8] S. A. Aljunid, Abdullah Mohd Zin, and Zarina Shukur, "A Study on the Program Comprehension and Debugging Processes of Novice Programmers," *J. Softw. Eng.*, vol. 6, no. 1, pp. 1–9, 2012.

[9] M. Hassan, "How do we Help Students 'See the Forest from the Trees?,'" in *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 2*, 2022.

[10] Z. Ahsan, U. Obaidellah, and M. Danaee, "Is Self-Rated Confidence a Predictor for Performance in Programming Comprehension Tasks?," *APSIPA Trans. Signal Inf. Process.*, vol. 11, no. 1, 2022.

[11] N. Al Madi and M. Zang, "Would a Rose by any Other Name Smell as Sweet? Examining the Cost of Similarity in Identifier Naming," in *The 33rd Psychology of Programming Interest Group (PPIG 2022)*, 2022.

[12] H. Eicken *et al.*, "Connecting Top-Down and Bottom-Up Approaches in Environmental Observing," *Bioscience*, vol. 71, no. 5, pp. 467–483, May 2021.

[13] S. Letovsky, "Cognitive processes in program comprehension," *J. Syst. Softw.*, vol. 7, no. 4, pp. 325–339, Dec. 1987.

[14] A. Fekete and Z. Porkoláb, "A comprehensive review on software comprehension models," *Ann. Math. Informaticae*, vol. 51, pp. 103–111, 2020.

[15] A. A. Shargabi, S. A. Aljunid, M. Annamalai, and A. M. Zin, "Performing Tasks Can Improve Program Comprehension Mental Model of Novice Developers," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020.

[16] P. Lima, J. Melegati, E. Gomes, N. S. Pereira, E. Guerra, and P. Meirelles, "CADV: A software visualization approach for code annotations distribution," *Inf. Softw. Technol.*, vol. 154, p. 107089, Feb. 2023.

[17] E. Fregnan, J. Fröhlich, D. Spadini, and A. Bacchelli, "Graph-based visualization of merge requests for code review," *J. Syst. Softw.*, vol. 195, p. 111506, Jan. 2023.

[18] Stephan Diehl, *Software Visualization - Visualizing the Structure, Behaviour, and Evolution of Software*. 2007.

[19] N. Chotisarn *et al.*, "A systematic literature review of modern software visualization," *J. Vis.*, vol. 23, no. 4, pp. 539–558, Aug. 2020.

[20] Azila Adnan and Muhamad F B Noor Hassim, "Infographics in Teaching and Learning: An Attention Grabber," in *International University Carnival on E-Learning (IUCEL) Proceedings 2022*, 2022.

[21] M. Dias, D. Orellana, S. Vidal, L. Merino, and A. Bergel, "Evaluating a Visual Approach for Understanding JavaScript Source Code," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020.

[22] M. Kargar, A. Isazadeh, and H. Izadkhah, "Improving the modularization quality of heterogeneous multi-programming software systems by unifying structural and semantic concepts," *J. Supercomput.*, vol. 76, no. 1, pp. 87–121, Jan. 2020.

[23] D. Limberger, W. Scheibel, J. van Dieken, and J. Döllner, "Procedural texture patterns for encoding changes in color in 2.5D treemap visualizations," *J. Vis.*, Oct. 2022.

[24] L. Bedu, O. Tinh, and F. Petrillo, "A Tertiary Systematic Literature Review on Software Visualization," in *2019 Working Conference on Software Visualization (VISSOFT)*, pp. 33–44, 2019.

[25] R. Ishizue, K. Sakamoto, H. Washizaki, and Y. Fukazawa, "PVC.js: visualizing C programs on web browsers for novices," *Heliyon*, vol. 6, no. 4, p. e03806, Apr. 2020.

[26] M. Mladenović, Ž. Žanko, and M. Aglić Čuvić, "The impact of using program visualization techniques on learning basic programming concepts at the K–12 level," *Comput. Appl. Eng. Educ.*, vol. 29, no. 1, 2021.

[27] M. Altherwi, "An empirical study of programming language effect on open source software development," in *Proceedings Companion of the 2019 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, 2019.

[28] Mohan Krishna Kagita and Li Xiujuan, "Machine Learning Techniques for Multimedia Communications in Business Marketing," *J. Mult. Log. Soft Comput.* , vol. 36, no. 1, pp. 151–167, 2021.

[29] H. He, "Understanding source code comments at large-scale," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019.

[30] S. Panthaplackel, J. J. Li, M. Gligoric, and R. J. Mooney, "Deep Just-In-Time Inconsistency Detection Between Comments and Source Code," *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 1, pp. 427–435, May 2021.

[31] X. Song, H. Sun, X. Wang, and J. Yan, "A Survey of Automatic Generation of Source Code Comments: Algorithms and Techniques," *IEEE Access*, vol. 7, pp. 111411–111428, 2019.

[32] J. Nielsen, J. Lewis, and C. Turner, "Determining Usability Test Sample Size," in *International Encyclopedia of Ergonomics and Human Factors, Second Edition - 3 Volume Set*, CRC Press, 2006.