Performing Tasks Can Improve Program Comprehension Mental Model of Novice Developers

An Empirical Approach

Amal A. Shargabi
Information Technology Department
Qassim University
Saudi Arabia
a.alshargabi@qu.edu.sa

Syed Ahmad Aljunid
Faculty of Computer and Mathematical Sciences
Universiti Teknologi MARA
Malaysia
aljunid@tmsk.uitm.edu.my

Muthukkaruppan Annamalai
Faculty of Computer and Mathematical Sciences
Universiti Teknologi MARA
Malaysia
mk@tmsk.uitm.edu.my

Abdullah Mohd Zin
Faculty of Information Science & Technology
Universiti Kebangsaan Malaysia
Malaysia
amzftsm@ukm.edu.my

ABSTRACT

Program comprehension is challenging for many novice developers. Literature indicates that program comprehension is greatly influenced by the specific purpose of reading a program, i.e., the task. However, the task has often been used in research as a measure for program comprehension. Our study takes an inverse approach to investigate the effect of using the task as a facilitator to improve novice developers program comprehension. To measure the effect, our previously published program comprehension mental model of novice developers was utilized. In a sense, the study provides an empirical evaluation of our proposed model in terms of its ability to capture the novice developer's mental model properly. The comprehensive experiment involved one hundred and seventy-eight (178) novice developers from three (3) universities and investigated the effect of six (6) tasks with difficulties ranked according to the cognitive categories of Revised Bloom Taxonomy. The results of the experiment confirmed that performing the tasks can improve program comprehension of novice developers. It demonstrated that different tasks improved different abstraction levels of the mental model and further indicated that higher cognitive category tasks improve program comprehension mental model at higher abstraction levels. The results also showed that the mental model we have proposed earlier is able to capture what novice developers know in response to the tasks they perform. The general implication of the study is that the tasks can be an effective tool for computing educators to incorporate program comprehension in programming courses, whereby these tasks need to be introduced in stages in the teaching of programming; starting initially from the lower cognitive categories' tasks such as Recall and culminating at the higher cognitive categories' tasks such as Modification by first taking into consideration the novices' programming levels.

CCS CONCEPTS

• Empirical studies • Experimentation • Software post-development issues • Computing education programs

KEYWORDS

Program Comprehension, novice developer, mental model, task

ACM Reference format:

Amal Shargabi, Syed Aljunid, Muthukkaruppan Annamalai, and Abdullah Zin. 2020. Performing Tasks Can Improve Program Comprehension Mental Model of Novice Developers: An Empirical Approach. In *Proceedings of 28th IEEE/ACM International Conference on Program Comprehension (ICPC '20). Seoul, Republic of Korea, 11 pages.* https://doi.org/10.1145/3387904.3389277

1 Introduction

Program comprehension (PC) is the process of understanding programs. It is a crucial component of successful programming, and a critical skill-set in software development and maintenance as programmers must understand source code before they can extend or maintain the software. Over the past decades, different program comprehension models have been proposed. These include four types: top-down [1, 2], bottom-up [3, 4], integrated [5], and opportunistic [6]. Despite the differences among these program comprehension models, most of them share the same basic three elements; the knowledge base, the assimilation process, the mental model, and a fourth related element, which is the external representation (Figure 1).

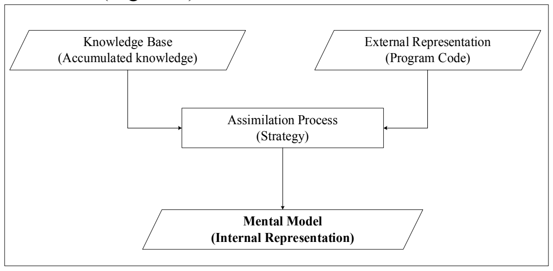


Figure 1: Elements of Program Comprehension Models (Inspired the work of Letovsky [7])

The knowledge base is the accumulated knowledge the programmer uses to understand the program code. Such knowledge includes: (1) the general programming knowledge, (2) the domain knowledge, and (3) language-related knowledge [7]. The assimilation process is the actual strategy that the programmer applies to comprehend the program code. The mental model is the programmer's internal representation of the program code under study [8] and typically described in program comprehension literature as having several levels of abstraction of the program code. The external representation is tied to the program code itself and other external documentation. Thus, the program comprehension model describes the processes (assimilation process) and information structures (knowledge base) in the programmer's head that are used to form the mental model.

Moreover, researchers acknowledged that the variety of the proposed program comprehension models is due to three main factors [9, 10] of who, what, and why: 1) who is trying to comprehend (the characteristics of the programmer), 2) what we are attempting to comprehend (the characteristics of the program), and 3) why we are attempting to comprehend (the tasks) [11]. These factors can affect some or all of the three elements of the program comprehension models. This study focuses on the third factor, the task, and the mental model element of program comprehension (Figure 2).

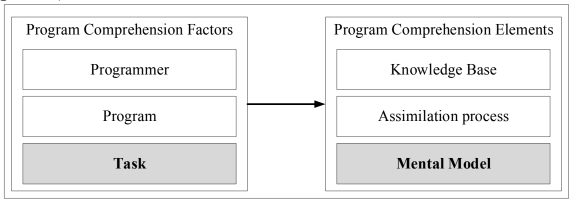


Figure 2: Scope of this Study

In the literature, tasks have widely been used as a measure for program comprehension of programmers [12]. The tasks used are often maintenance tasks such as debugging [13, 14] and enhancement [13]. Programmers are asked to perform such tasks on source code, and the successful performance of the task is an indication of their understanding of the code. On the contrary, this study applies the inversion approach of viewing tasks as a means to improve program comprehension rather than the traditional approach of measuring program comprehension using tasks.

Although novices have weaknesses in the three core elements of the program comprehension model, i.e., the knowledge base, the mental model, and the assimilation process, we focus on the mental model element only. This is because the knowledge base and the assimilation process of novices are based on external parameters that are difficult to be developed within a short time. Both of them are essentially based on exposure, domain knowledge, familiarity, and expertise, which all expand over time. Although we acknowledge that both complementary factors assimilation process and mental model mutually influence each other and should not be separated, we argue there is a special case for novices. In the case of experts, comprehension strategies are more emphasized in order to derive guidelines for new supporting tools to expedite the program comprehension process (e.g., [15-21]). On the other hand, there is a dire need to focus first on the mental model for novices rather than strategies. Consequently, a previously published program comprehension mental model of novice developers [22] was used and evaluated in this study. A summary of this model is presented in Section 3. We used an empirical approach to investigate and compare the effect of six tasks (focused on the categories of Revised Bloom taxonomy) on novice developers' program comprehension mental model through a comprehensive experiment involving one hundred and seventy-eight (178) novice developers, from three (3) different universities.

The remainder of the paper is organized as follows: Section 2 elaborates on the motivation of this study. Section 3 provides an overview of the program comprehension mental model used in the experiment. Section 4 gives a thorough description of the methodology used in the experiment, followed by Section 5, which presents the results of the experiment. Section 6 provides a summary of the findings as well as a discussion of the outcomes of the study, and Section 7 discusses the threats of the validity of the conducted experiment. Finally, Section 8 concludes the paper by describing the possible implications of the conducted study in programming pedagogy, limitations, and future directions for other researchers.

2 Motivation

Novices face great difficulty in program comprehension [23-25]. In literature, only a few studies investigated the effect of tasks on the improvement of program comprehension mental models. These studies, except Burkhardt et al. [26], did not compare between the different tasks regarding their effectiveness in improving program comprehension; rather, they investigated the effect of a single task on program comprehension. Furthermore, these studies investigated a limited number of tasks, and some of them studied a single task only. For example, Du Bois et al. [27] studied the refactoring task alone, while Pennington [28] investigated modification tasks only. And most of these studies were not meant for novices, but experts, such as the works of von Mayrhauser and her colleagues [5, 29], which utilized some tasks like debugging and enhancement. Another significant point is that these tasks were studied individually, and thus no comparisons were conducted among them. In addition, most of these studies were mainly based on Pennington's mental model [4] of experts. In fact, most program comprehension models, and their related mental models were constructed for experts. These models are typically grounded in experimental studies, mostly using large scale programs of thousands of lines of code (LOC). For example, in the work of von Mayrhauser's model [5], the sizes of the programs used were more than 40,000 LOC. Thus, these models may not be applicable for novices. There is one mental model proposed by Schulte [30] for novices; however, the novices in the context of Schulte's model are novice programming instructors rather than novice programming students themselves. Also, applying the Schulte's model to measure program comprehension in experiments is challenging as there are too few examples of how to use the model to measure program comprehension.

In a previous study [22], we proposed a new conceptualization for the novice developer's mental model for program comprehension. This study makes use of this model and conducted an experiment on novice developers to achieve two goals: 1) To investigate and compare the effect of using tasks of different cognitive requirements on the program comprehension mental model of novices, 2) To empirically evaluate the appropriateness of our proposed model in describing the mental model of novices. The outcome of the evaluated model can serve as a practical guide for educators to incorporate comprehension tasks in programming pedagogy and provide valuable reusable design and instruments for future experiments involving novices' in relation to program comprehension experiments.

The following section gives an overview of our proposed model that is utilized in this study.

3 An Overview of the Proposed Program Comprehension Mental Model of Novices

Figure 3 shows the model [22], which is made up of five hierarchal levels of abstraction: *Statement*, *Block*, *Module*, *Program*, and *Domain*. Statement Level is the lowest abstraction level, while Domain Level is the highest. Every abstraction level involves certain information categories. However, some information categories, such as control flow and data flow, appear at multiple abstraction levels but at different difficulties. As we move from lower abstraction levels into higher abstraction levels, lines of code, i.e., code size increases.

The proposed model is founded on two prior models, Pennington's model [4] and von Mayrhauser's model [31] that were refined to novices' context. Both these models were refined to consider the limited abstraction capabilities of novices as compared to experts; in other words, the line of codes that novices can comprehend are much smaller than those of the experts. Thus, the abstraction levels of this model were conceptualized to consider the size of the code and start with the lowest level of abstraction of a single statement, i.e., Statement, to the abstraction of a sequence of statements, i.e., Block, to the abstraction of the individual module(s), i.e., Module, to the abstraction of the entire program that contains multiple interacting modules, i.e., Program, and eventually ends with the highest abstraction that links the entire program with the domain knowledge, i.e., Domain.

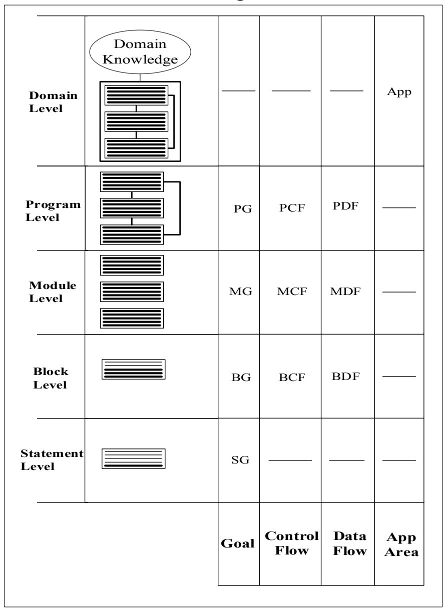


Figure 3: The Proposed Novices' Program Comprehension Mental Model (Adapted from our previous study [22])

The proposed model is founded on two prior models, Pennington's model [4] and von Mayrhauser's model [31] that were refined to novices' context. Both these models were refined to consider the limited abstraction capabilities of novices as compared to experts; in other words, the line of codes that novices can comprehend are much smaller than those of the experts. Thus, the abstraction levels of this model were conceptualized to consider the size of the code and start with the lowest level of abstraction of a single statement, i.e., Statement, to the abstraction of a sequence of statements, i.e., Block, to the abstraction of the individual module(s), i.e., Module, to the abstraction of the entire program that contains multiple interacting modules, i.e., Program, and eventually ends with the highest abstraction that links the entire program with the domain knowledge, i.e., Domain.

Statement Level contains only one information category, the Statement Goal (SG), which is the goal of individual statements (syntactic and semantic meaning). This information category is mapped to the elementary operation in Pennington's and von Mayrhauser's models. Since this level contains only one line of code, it does not have a control flow or data flow information categories. Next, *Block Level* contains three types of information categories: Block Goal (BG), Block Data Flow (BDF), and Block Control Flow (BCF). BDF is associated with the changes of data objects, e.g., variables within the block, while BCF is associated with the sequence of execution within the block. The third level, *Module Leve*l, contains three types of information categories:

Module Goal, Module Data Flow (MDF), and Module Control Flow (MCF). *Program Level* involves three information categories: Program Goal (PG), Program Data Flow (PDF). Finally, the *Domain Level* is aligned to the context of the program. It answers the question of where else the current program can be applied [22]. The information category involved in this level is the Application Area of the program (App). It requires domain knowledge of the problem.

The proposed model is used to develop the program comprehension questions of the experiment explained in the following section.

4 Experiment

This study is to investigate the effect of the tasks on the novices' program comprehension mental model (PC MM). The studied tasks are identified based on the results of a previous study of ours [32]. In that study, we identified a list of ranked tasks that potentially can improve the novice developer's program comprehension mental model. The ranking of the tasks was based on the instructors' perspective, i.e., a survey at several universities. The tasks were ranked within homogenous categories corresponding to the cognitive categories of the Revised Bloom's Taxonomy they belong to. Table 1 shows the ranked tasks.

Table 1: Task Ranking based on Instructors' Perspectives [32]

Rank	Task	Cognitive category
1	**Recall**	Remember
2	Line Documentation	
1	**Representation**	Understand
2	Summarization	
1	**Renaming**	Apply
1	**Tracing**	Analyze
2	Debugging	
3	Search	
4	Reordering	
1	**Comparison**	Evaluate
1	**Modification**	Create
2	Extension	
3	Reuse	
4	Restructuring	

The tasks to be tested experimentally are six tasks, which are the first ranked task from each cognitive category. Comparisons are made between tasks in adjacent Bloom categories, as will be explained in Section 4.1. Tasks are not compared within each cognitive category as two categories include only one task, i.e., Renaming and Comparison, and thus no comparisons can be made. The full definitions and examples of all tasks are available in our previous study [32]. The design and analysis methodologies employed are based on Wohlin et al. [33] entitled *Experimentation in Software Engineering*, which is a key book in empirical software engineering studies.

4.1 Scoping

This experiment validates the effect of six tasks on the PC MM abstraction levels and information categories of novices. In particular, comparisons are made between the tasks in adjacent Bloom categories as follows: 1) Recall task (Remember category) is compared to Representation task (Understand category), 2) Renaming task (Apply category) is compared to Tracing task (Analyze category), and 3) Comparison task (Evaluate category) is compared to Modification task (Create category).

4.1 Planning

4.2.1 Context Selection. The experiment is executed in three universities. To ensure a common basis for all respondents in programming/program comprehension background, a dedicated workshop, namely, Program Comprehension Workshop is conducted with arrangements with the respective faculties at these universities. The workshop agenda and timing are presented later in the operation section (Table 6).

4.2.2 Selection of Subjects. The subjects are chosen from three universities based on convenience sampling. They are semester four CS/IT students and reachable by the researchers. The sample size required in each university is 8-10 subjects per group, so that the total subjects from the three universities in each group are around 30, which is a reasonable number to make conclusions.

4.2.3 Variables Selection. The experimental variables include one independent variable, the type of task (Task), and a set of dependent variables based on the proposed model (Section 3). These include the overall program comprehension mental model (PC MM), the abstraction levels of the mental model, and the information categories of these levels. This experiment focuses on the first three abstraction levels, Statement Level (SL), Block Level (BL), and the Module Level (ML), and the corresponding information categories of these three levels which are: Statement Goal (SG), Block Goal (BG), Block Control Flow (BCF), Block Data Flow (BDF), Module Goal (MG), Module Control Flow (MCF), and Module Data Flow (MDF).

The independent variable, Task, is measured using correctness measure. For each task, three questions are developed, and each question is evaluated on a four-point scale to assess to what degree it is logically correct based on specific criteria. A scale for correctness is defined as follows. 1) Correct- the idea is logically right and performed without errors (10 marks); 2) Minor errors- the idea is logically right but performed with minor errors (8 marks); 3) Major errors- the idea is logically right but performed with major errors (5 marks); and 4) Incorrect- the idea is logically wrong or no answer (0 marks). Using the above scale, a scoring scale is to be developed for each task type. Table 2 shows a sample of the scoring scale for Modification task.

Table 2: Sample of Task Scoring Scales

Task	Criteria	Rating scales / Grading Policy
Modification	Logic	**Correct**: Idea is logically right but implemented without syntactic errors (10 marks)
	Syntactic errors	**Minor errors**: Idea is logically right but implemented with minor syntactic errors (8 marks)
		Major errors: Idea is logically right but implemented with major syntactic errors (5 marks)
		Incorrect: Idea is logically wrong or no answer (0 marks)

Each type of task, e.g., Recall, is measured by three questions, one for each method. The total score of the Task variable is the mean score of the three task questions (in ratio scale). This approach is called a holistic rubric, in which a single score, i.e., usually on a 1 to 4 or 1 to 6-point scale, is assigned based on an overall judgment of the student answer. It is commonly used in measuring tasks in program comprehension and computer science education research [34]. As an example, the refactoring tasks were measured in the study of Du Bois et al. [27] using four scales that are: wrong answer, right idea (5), almost correct (8), and correct (10).

The dependent variable, PC MM, is calculated as the mean score of its abstraction levels, which are calculated as the mean score of its information categories. The information categories of the PC MM's abstraction levels are measured using a correctness measure. Three multiple-choice questions are developed for each information category made up of the abstraction levels. The sample questions are shown in Table 4. Each question is evaluated on a binary scale; that is, an incorrect answer is rated as 0 and correct answer as 1. The total score of each abstraction level is the mean score of the information categories' questions made up that level, i.e., nominal scale, and the total score of PC MM is the mean of all abstraction levels' score.

4.2.4 Choice of Design Type. Subjects are divided into six groups, G1 to G6, corresponding to the six tested tasks. i.e., each group works on one task only.

As the tasks are at different levels of difficulty and to make sure that each group is able to perform the assigned task, the following method is used for group assignment. The subjects are first divided into three clusters: weak, moderate, and strong. The criterion used to cluster subjects is their grades of two previous programming courses, i.e., the average GPA of the two courses, which was gathered prior to the experiment. The weak subjects are given low cognitive category tasks, i.e., Recall and Representation. The moderate subjects are given medium cognitive category tasks, i.e., Renaming and Tracing. Strong subjects are given high cognitive category tasks, i.e., Comparison and Modification. Random blocking is then applied. Accordingly, G1 and G2 are randomly formed from weak subjects and given Recall and Representation, respectively. G3 and G4 are randomly formed from moderate students and given Renaming and Tracing, respectively. G5 and G6 are randomly formed from the strong students and given Comparison and Modification, respectively. That is, Recall group (G1) is compared with Representation group (G2), Renaming group (G3) is compared to Tracing group (G4), and Comparison group (G5) is compared to Modification group (G6). Therefore, a between-group design (two-group) is selected to investigate the difference between every two comparative groups. To measure the development of performing tasks within each group, a pre-test-post-test design is chosen. This design type, i.e., the two-group pre-test-post-test design, is used in prior program comprehension empirical studies such as Burkhardt et al.[26].

Specifically, the design contains three steps a pre-test, a treatment, i.e., task, and a post-test (Table 3). Three codes are used method1, method2, and method3 (Appendix). The pre-test is given on the three methods and referred to as P0M1 and P0M1, and P0M3, respectively. Similarly, the post-test is also given on the three methods and referred to as P1M1 and P1M2, P1M3. The pre-tests measure the PC MM abstraction levels and information categories of the groups before the task performance, and the post-tests measure the PC MM after the task performance. By studying the differences between the post-test and pre-test scores of the comparable task groups, conclusions can be drawn with respect to the effect of the tasks on the PC MM abstraction levels and information categories. For the treatment step, each group performs three tasks corresponding to the three methods. The three tasks are of the same type for each group. For example, Recall group (G1) performs three recall tasks, one task on each of the three methods.

Table 3: Experimental Design

Pre-test	**Pre-test on method1, method2, and method3**
Treatment	**Task on method1, method2, and method3** Recall (G1) vs. Representation (G2) Renaming (G3) vs. Tracing (G4) Comparison (G5) vs. Modification (G6)
Post-test	**Post-test on method1, method2, and method3**

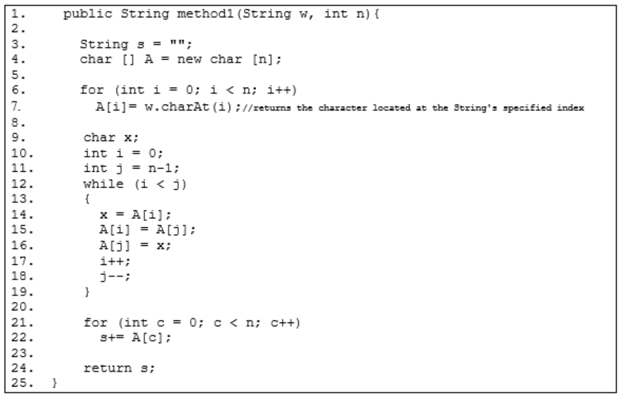
4.2.5 Hypotheses Formation.

Hypothesis H01-There is no difference in program comprehension mental model development within all groups after performing the assigned task.

Hypothesis H02-There is no difference in program comprehension mental model development between every two comparable groups.

4.2.6 Instrumentation Preparation. There are three types of instruments: programs, pre-test/post-test, and tasks.

4.2.6.1 Programs. Three codes are used, method1, method2, and method3. *method1* is to reverses a string word using the original array (LOC =25) and was adapted from the study of Lister et al. [35], *method2* is to find the number of common elements between two arrays (LOC =23) [35], and *method3* is to display bar chart of grades stored in a two- dimensional array (LOC =22) and was adapted from the Java book of Deitel and Deitel [36]. Method1 is shown in Figure 4 as a sample.



```
public String method1(String w, int n){

   String s = "";
   char [] A = new char [n];

   for (int i = 0; i < n; i++)
     A[i]= w.charAt(i);//returns the character located at the String's specified index

    char x;
    int i = 0;
    int j = n-1;
    while (i < j)
    {
      x = A[i];
      A[i] = A[j];
      A[j] = x;
      i++;
      j--;
    }

    for (int c = 0; c < n; c++)
      s+= A[c];

    return s;
 }
```

Figure 4: Method1 Code (Adapted from [35])

4.2.6.2 Pre-test/post-test. The pre-test and post-test contain questions corresponding to the seven information categories that made up the three abstraction levels examined. For each information category, three questions are developed, one for each method. The pre-test and post-test questions are identical. Examples of questions are shown in Table 4.

Table 4: Example of Pre-Test /Post-Test Questions of method1

AL	IC	Question
SL	**G**	After the first `for` statement *(lines 6-7)* is executed, `A[2]` contains the value: (a) `M` (c) `L` (b) `A` (d) `Y`

BL	**G**	In the block below *(lines 14-16)*
		`x = A[i];`
		`A[i] = A[j];`
		`A[j] = x;` What does the above block do?
		(a) Exchanges the value `A[i]` with the value of `A[j]`
		(b) Exchanges the value `x` with the value of `A[i]`
		(c) Exchanges the value `x` with the value of `A[j]`
		(d) Compares the value `A[i]` with the value of `A[j]`

	CF	In the block below *(lines 14-16)*
		`x = A[i];`
		`A[i] = A[j];`
		`A[j] = x;`
		(a) `A[i]` is assigned to `x` before it is replaced by `A[j]`
		(b) `A[j]` is assigned to `x` before it is replaced by `A[i]`
		(c) `x` is assigned to `A[i]` before it is replaced by `A[j]`
		(d) `x` is assigned to `A[j]` before it is replaced by `A[i]`

ML	**G**	The main purpose of `method1` is to: (a) Reverse a word using the original array (b) Reverse words of a sentence using the original array (c) Reverse a word using a new array (d) Reverse words of a sentence using a new array
	DF	When the `while` loop is exited, the value of `s` will be: (a) MALAYSIA#1 (c) 1#AISYALAM (b) " " (d) #1ALAMYSIA

Note: AL: Abstraction Level, IC: Information Category, SL: Statement Level, BL: Block Level, ML: Module Level, G: Goal, CF: Control Flow, DF: Data Flow

4.2.6.3 Task Questions. For each method, six questions are prepared corresponding to the six types of tasks being studied. For example, for method1, six task variants are developed: T1M1, T2M1, T3M1, T4M1, T5M1, and T6M1. Sample task questions on method1 are shown in Table 5.

Table 5: Sample Task Questions on method1

Task	Question
Recall	Complete the missing statements in method1.
Representation	Translate method1 from Java code into its corresponding flowchart.
Renaming	Use meaningful names for s, x, method1.
Tracing	What is the return value of method1?
Comparison	Explain the semantic differences between the two methods.
Modification	Modify the code so that it accepts a string of alphabets and numbers and reverses *only the alphabets and ignore numbers*. That is, if w = "123MALAYSIA123", method1 returns the value "AISYALAM".

4.3 Operation

The experiment took place at three Malaysian universities, Universiti Teknologi MARA (UiTM), Selangor state, Sultan Idris Education University (UPSI), Pahang state, and Universiti Utara Malaysia, (UUM), Kedah state. The experiment was carried out in the three universities as a one-day workshop entitled Program Comprehension Workshop. The workshop agenda is shown in Table 6.

Table 6: Workshop Program

Program	Time (Min)
Workshop Overview	15
Program Comprehension - Part 1 + Break	45 + 20
Programming Competency Test - Briefing	**10**
Programming Competency Test + Break	**120** + 60
Program Comprehension - Part 2	180

The workshop involves a Programming Competency Test, which is the actual experiment. Besides the test, the workshop includes a lecture on program comprehension basics (Program Comprehension – Part1) and engages subjects in practicing some program comprehension exercises (Program Comprehension – Part 2). The program competency test contains three sessions. The sessions and timing of the test are shown in Table 7.

Table 7: Programming Competency Test Timing

Program		Time (Min)
The briefing (programming competency test)		10
Session 1 (Pre-test)	P0M1(Pre-test on method 1)	10
	P0M2 (Pre-test on method 2)	10
	P0M3 (Pre-test on method 3)	15
Session 2 (Treatment)	TnM1 (Task on method 1)	15
	TnM2 (Task on method 2)	15
	TnM3 (Task on method 3)	15
Session 2 (Post-test)	P1M1 (Post-test on method 1)	10
	P1M2 (Post-test on method 2)	10
	P1M3 (Post-test on method 3)	15

5 Results

This section presents a statistical analysis of the gathered data.

5.1 Descriptive Statistics

Table 8 shows the number of subjects in each group per university. In total, 178 subjects take part in the experiment, 58 are from UiTM, 64 from UPSI, and 56 from UUM.

To examine whether the subjects improved after the assigned tasks, Table 9 shows the mean scores for the Pre-test Post-test for all groups in terms of the overall program comprehension, and the three examined abstraction levels. By investigating the PC MM column of both pre-test and post-test in Table 9, it can be seen that subjects in the six groups score higher in the post-test than the pre-test. For example, the mean score of Renaming group increases from 44.65 to 57.83 after performing Renaming tasks. It can also be seen that some tasks have a greater effect than others on PC MM. For example, Tracing tasks have a greater effect compared to Renaming tasks.

Table 8: Number of Subjects in Each Group per University

Cognitive Category	Task	University			Total
		UiTM	UPSI	UUM	
Remember	Recall	10	10	11	**31**
Understand	Representation	11	10	11	**32**
Apply	Renaming	9	11	8	**28**
Analyze	Tracing	9	11	8	**28**
Evaluate	Comparison	9	11	9	**29**
Create	Modification	10	11	9	**30**
Total		58	64	56	178

Table 9: Scores of the Abstraction Levels and the Overall PC MM

Cognitive Category	Task	Pre-test score				Post-test score			
		SL	BL	ML	PC MM	SL	BL	ML	PC MM
Remember	Recall	37.65	46.52	32.03	38.73	74.23	57.84	37.48	56.52
Understand	Representation	40.53	39.44	34.84	38.27	45.72	58.78	43.56	49.35
Apply	Renaming	50.00	48.71	35.25	44.65	58.36	65.50	49.64	57.83
Analyze	Tracing	52.46	54.00	40.00	48.82	61.89	76.39	61.96	66.75
Evaluate	Comparison	62.17	60.28	42.76	55.07	67.86	79.45	69.07	72.13
Create	Modification	63.47	59.37	40.17	54.33	72.30	79.77	65.63	72.57

SL: Statement Level, BL: Block Level, ML: Module Level, PC MM: Program Comprehension Mental Model

Similarly, examining the Statement, Block, and Module levels' columns of both pre-test and post-test shows that subjects in the six groups also score higher in the post-test than the pre-test. For example, the mean score of Renaming group in Statement, Block, and Module levels increase from 50.00, 48.71, 35.25 to 58.36, 65.50, and 49.64, respectively. However, some tasks have a greater effect than others do. The improvement in Statement Level is greater for Recall group as compared to Representation group but not the improvement in Block and Module levels.

The descriptive statistics shown in Table 9 indicate that there are differences between the comparable tasks in terms of the overall program comprehension mental model and its abstraction levels. However, these statistics do not tell whether these differences are *significant or not*. Thus, the hypotheses test is conducted in the following section.

5.2 Hypotheses Tests

For testing hypothesis H01, Paired Samples T-Test for dependent samples is used, because the data collected for this hypothesis is within-subjects, i.e., post-test scores are compared to pre-test scores of subjects within the same group [37]. For testing hypothesis H02, the appropriate test is Independent Samples T-Test for independent samples [37]. This is because the data collected for this hypothesis is between subjects, i.e., the difference between post-test scores and pre-test scores of subjects are compared between the two comparable groups.

5.2.1 Hypothesis H01- There is no difference in program comprehension mental model development within all groups after performing the assigned task. Table 10 shows separately the results of testing hypothesis H01 for each group (G1-G6) using Paired-Samples T-Test. The table shows the t value, df value, and the associated p-value for the dependent variables. By examining the last column of Table 10, one can see that all groups achieve significant improvement in the overall program comprehension mental model. For the improvement of the three abstraction levels, the results are as follows. First, Recall group is the only group that achieves significant improvement in the Statement Level (p-value=.000). The other five groups do not show a significant difference between the pre-test and the post-test. Second, all groups achieve significant improvement for the Block Level (p-value=.004 for Recall groups and .000 for the other five groups). Third, G2-G6 achieved significant improvement for the Module Level (p-value=.013, .004, .001, .000, .000, respectively). To further investigate the effect of the tasks on the abstraction levels, it is essential to examine what information categories made up the abstraction levels improved after performing each task. Table 11 shows the results of Paired Groups T-Test for the information categories made up of each abstraction level for the six groups. Based on Table 11, it can be seen that all groups showed significant improvement in three or more information categories. For example, Representation group achieved a significant improvement in the Block Goal, Block Control Flow, and Module Control Flow information categories. As such, hypothesis H01can be rejected. A discussion on these results is presented in Section 6.1.

5.2.2 Hypothesis H02- There is no difference in program comprehension mental model development between every two comparable groups. Table 12 shows the results of testing hypothesis H02 for G1 vs. G2, G3 vs. G4, and G5 vs. G6. Due to the limited space for writing, only p-values are presented. By examining the p-values column in Table 12, one can see the following results. There is a significant difference between Recall and Representation groups in the improvement of Statement Level only. The former shows greater improvement in this level after performing Recall tasks as compared to the latter, which performed Representation tasks. For the other two pairs, the differences are not significant. To get a closer look at the potential differences between the comparable task pairs, Independent Samples T-Test is run for the information categories made up of the abstraction levels in each task pair. The test results are shown in Table 13, which shows the p-value for the information categories of each task pair. Other statistics, i.e., T and df, are not presented to the limited space of writing.

Table 10: T-Test Results of the Abstraction Levels (Post-test vs. Pre-test)

Task	Post_SL- Pre_SL			Post_BL- Pre_BL			Post_ML- Pre_ML			Post_PC MM- Pre_PC MM		
	t	df	P	t	df	P	t	df	P	t	df	P
Recall (G1)	-5.516	30	**.000**	-3.101	30	**.004**	-1.514	30	.140	-6.933	30	**.000**
Representation (G2)	-0.733	31	.469	-5.445	31	**.000**	-2.625	31	**.013**	-3.733	31	**.001**
Renaming (G3)	-1.758	27	.090	-6.128	27	**.000**	-3.102	27	**.004**	-5.195	27	**.000**
Tracing (G4)	-1.740	27	.093	-7.513	27	**.000**	-3.710	27	**.001**	-5.661	27	**.000**
Comparison (G5)	-1.080	28	.289	-5.061	28	**.000**	-5.875	28	**.000**	-5.563	28	**.000**
Modification (G6)	-1.847	29	.075	-6.403	29	**.000**	-6.237	29	**.000**	-8.380	29	**.000**

Table 11: T-Test Results of the Information Categories (Post-test vs. Pre-test)

	SL	BL			ML		
Task	Post_SG-Pre_SG	Post_BG-Pre_BG	Post_BCF-Pre_BCF	Post_BDF-Pre_BDF	Post_MG-Pre_MG	Post_MCF-Pre_MCF	Post_MDF-Pre_MDF
	P	P	P	P	P	P	P
Recall (G1)	**.000**	.344	**.022**	.110	.346	**.011**	.389
Representation (G2)	.469	**.000**	**.000**	.412	.870	**.002**	.708
Renaming (G3)	.090	**.047**	**.025**	**.000**	.185	**.001**	.574
Tracing (G4)	.093	**.000**	**.028**	**.004**	.246	**.007**	**.001**
Comparison (G5)	.289	**.001**	.154	**.000**	**.016**	**.001**	**.000**
Modification (G6)	.075	**.000**	.479	**.013**	**.001**	**.003**	**.000**

Table 12: Results for PC MM Improvement in terms of Abstraction Levels

Task	Diff_SL			Diff_BL			Diff_ML			Diff_PC MM		
	T	df	P	t	df	P	T	df	P	t	df	P
Recall (G1) vs. Representation (G2)	3.231	61	**.002**	-1.582	61	.119	-.673	61	.504	1.703	61	.094
Renaming (G3) vs. Tracing (G4)	-.175	54	.862	-1.387	54	.171	-1.022	54	.311	-1.171	54	.247
Comparison (G5) vs. Modification(G6)	-.438	57	.663	-.231	57	.818	.144	57	.886	-.314	57	.754

Table 13: Results for PC MM Improvement in terms of Information Categories

	SL	BL			ML		
Task	Diff_SG	Diff_BG	Diff_BCF	Diff_BDF	Diff_MG	Diff_MCF	Diff_MDF
	P	P	P	P	P	P	P
Recall (G1) vs. Representation (G2)	**.002**	**.020**	.168	.490	.427	.548	.688
Renaming (G3) vs. Tracing (G4)	.862	**.018**	.799	.568	.996	.746	**.012**
Comparison (G5) vs. Modification(G6)	.663	**.002**	.402	.157	.664	.244	.661

Note: SL: Statement Level, BL: Block Level, ML: Module Level, SG: Statement Goal, BG: Block Goal, BCF: Block Control Flow, BDF: Block Data Flow, MG: Module Goal, MCF: Module Control Flow, MDF: Module Data Flow

From Table 13, it can be seen that there is a significant difference between the comparable task pairs in at least one information category. For example, there is a significant difference between Renaming and Tracing in the Block Goal, and Module Data Flow information categories as the p-values are 0.018 and 0.012, respectively. This implies that there are significant differences between the two tasks at the Block and Module abstraction levels, even the results shown in Table 12, does not show these differences. As such, hypothesis H02 can be rejected. A discussion on these results is presented in Section 6.2.

6 Findings and Discussion

6.1 The Effect of the Tasks on the Program Comprehension Mental Mode

The results reported in the previous section showed that *the different tasks improve different abstraction levels and different information categories depending on the nature of the task.* The six tested tasks are discussed below. The Recall task is to remember the program code and to write it statement by statement. It requires the subjects to understand each statement and capture the flow of these statements. Thus, this task improves the Statement Goal, Block Control Flow, and Module Control Flow information categories. The Representation task is to provide flow chart drawings for the program code. It requires the subjects to understand the flow of the code, such as the iterations and condition, and understand the goal of the code or part of it. Thus, this task improves Block Goal, Block Control Flow, and Module Control Flow information categories. The Renaming task is to label program code identifiers with more meaningful names. It requires the subjects to understand the program code in terms of the data flow, the control flow, as well as the goal of the code or part of it. Thus, this task improves the Block Goal, Block Data Flow, Block Control Flow, and Module Control Flow information categories. The Tracing task is to follow and aggregate the different occurrence of the traced variable(s) in order to find the final output. It requires the subjects to understand the code in terms of the data flow, the control flow, as well as the goal of the code or part of the code. Thus, this task improves the Block Goal, Block Control Flow, Block Data Flow, Module Control Flow, and Module Control Flow information categories. The Comparison task is to Evaluate two alternative pieces of codes of common purpose and in terms of syntactic and semantic differences. It requires the subjects to understand the code in terms of the data flow, the control flow, as well as the goal of the code. Thus, this task improves the Block Goal, Block Data Flow, Module Goal, Module Control Flow, and Module Control Flow information categories. The Modification task is to build up their own codes from an existing code. It requires the subjects to understand the code in terms of the data flow, the control flow, as well as the goal of the code. Thus, this task improves the Block Goal, Block Data Flow, Module Goal, Module Control Flow, and Module Data Flow information categories.

6.2 The Differences between the Tasks Regarding Their Effect on the Program Comprehension Mental Model

Based on the results presented in the previous section, we can conclude that *higher cognitive categories tasks can improve higher abstraction levels in one or more information categories.* The details are as follows. Representation task belongs to Understand cognitive category, which is higher than Remember in which Recall belongs to. Representation contributes more to the improvement of the program comprehension mental model as compared to Recall. The improvement is in terms of Block Goal, the goal of parts of the code. Tracing belongs to Analyze cognitive category, which is higher than Renaming in which Apply belongs to. Tracing contributes more to the improvement of program comprehension as compared to Renaming. The improvement is in terms of Block Goal, the goal of parts of the code, and Module Control Flow, Module Data Flow, the flow of data. Modification belongs to Create cognitive category, which is higher than Comparison in which Evaluate belongs to. Modification contributes more to the improvement of program comprehension as compared to Comparison. The improvement is in terms of Block Goal, the goal of parts of the code.

6.3 The Appropriateness of the Proposed Model in Capturing the Improvement of Novices' Mental Model

Based on the results and findings of the experiment, the effect of the tasks on the PC MM of novices can be graphically represented as shown in Figure 5.

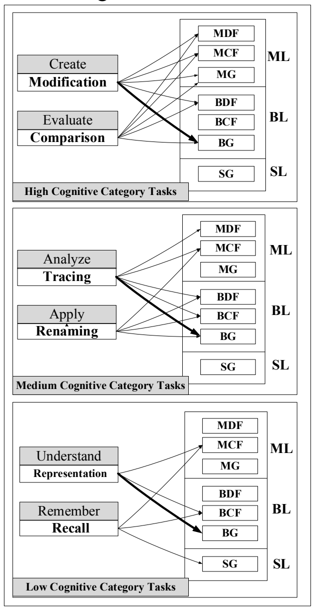


Figure 5: The Effect of the Tasks on Abstraction Levels and Information Categories of Novices' PC MM

The figure shows the comparable tasks from the adjacent cognitive categories and their effects on the development of novices' program comprehension. The directed lines in this figure show the information categories improved as a result of performing certain tasks. The bolded directed lines indicate that task that is more effective in developing the associated information category as compared to the other comparable task. For example, Tracing contributes more to the improvement of program comprehension as compared to Renaming. The improvement is in terms of Block Goal, the goal of parts of the code, and Module Control Flow, Module Data Flow, the flow of data. We believe that the conceptualization of our model is more representative of novices' mental model as compared to previous models since novices' struggle in comprehending larger size programs. Among the program comprehension issues novices' have, as indicated by Lahtinen et al. [38], are "understanding how to design a program to solve a certain task", "dividing functionality into procedures", and "finding bugs from their own programs", which all exist when novices must understand larger abstractions of bigger programs instead of just some details. Novices also may know the syntactic and the semantics of individual statements, but yet may not know how to combine them into valid programs [38].

The description of our mental model emphasizes the relation between the code size, i.e., program elements and the abstraction level required. This relation is reflected by the progression from the understanding of one line of code in the statement level, to the understanding of the entire program, in the program level, and its application, in the domain level. Unsurprisingly, novices have problems in developing a proper program comprehension situation level of Pennington's and Von Mayrhauser's models [5]. Thus, it is vital for them to describe their mental model as clear and distinct abstraction levels and information categories. As an example, the Tracing task was tested in the literature to be effective in the development of the situation level, but not the program level of Pennington's model [4]. In our model, the effect of the tracing task on the mental model can be viewed in more detail. The model shows that this task is effective in the development of control flow, which is part of the program level of Pennington's model. It also shows that it does not improve all information categories of the situation level. As an instance, the task does not improve the module goal, which is a central part of Pennington's situation level.

7 Threats to Validity

There are different classification schemes for different types of threats to validity. Campbell et al. [39] defined two types, threats to internal and external validity. Cook and Campbell [40] extended the list to four types: conclusion, internal, construct, and external validity. More recently, Siegmund [41] provided a comprehensive list of confounding parameters specifically for program comprehension experiments and included techniques to control them. Based on her list, the confounding parameters can be divided into two categories: personal and experimental. Threats to the validity of the conducted experiment in this study were based on Sigmund's list [41]. The most related confounding parameters are described below.

7.1 Personal Parameters

Education is controlled by keeping it constant. To measure it, subjects are selected from the same course. Thus, they have comparable knowledge. *Programming experience* is controlled by applying random blocking. To measure it, the average of two completed programming courses, i.e., GPA, is calculated and used as a base for assigning the subjects to the different groups. *Motivation* is controlled by keeping it constant. To ensure those subjects are motivated, the experiment is conducted under a programming competency test umbrella. Conducting the experiment within a formal workshop improves motivation and commitment as the test is given after a nice talk and engagement of subjects. Also, three rewards were also offered for the three best-performing subjects.

7.2 Experimental Parameters

Learning effects are controlled as a training, i.e., task briefing is given before the experiment. *Evaluation apprehension* is controlled as the grouping is implicit so that the subjects do not know that they are in different groups. *Study-object coverage* is controlled by keeping it constant. Subjects who do not complete the test, i.e., pre-test, tasks, and post-test or cannot answer the tasks, i.e., total task score is less than 50, are excluded from the analysis. *Instrumentation effect* is avoided by using the same code for all subjects and comparable tasks for the comparable groups. *Mono-operation bias* is controlled by using three questions for each task and three questions for each information category. To keep *Rosenthal constant*, a briefing is given to all groups by the same experimenter prior to the programming competency test. Also, the experiment is conducted in one hall. *Tasks* of the same cognitive category are compared. This ensures the task questions' difficulty levels are comparable. Moreover, there may appear some arguments that there is a necessity for a control group in the experiment. However, in the previous studies of program comprehension, there have been a few accepted and well-established works which did not apply control groups such as Burkhardt et al. [26]. In Burkhardt study [26], which was published in the Empirical Software Engineering journal, two groups were compared, Documentation and Reuse groups without including a control group. Moreover, in the context of this study, no control group is included due to the limited availability of sample respondents fitting the experimental criteria and also due to the large number of groups needed to be formed at each experimental location, i.e., six groups in three universities.

8 Conclusion

8.1 Implications of Findings

The findings derived from the experiment indicated that, based on the task nature, different aspects, i.e., the information categories of the mental model, are developed, and some tasks are more effective than others (Table 14). The implication of the findings is that computing educators can apply the tasks accordingly by taking the consideration of the students' programming levels and the information categories they want to improve. For example, to help *weak novices, i.e., low programming level novices,* improve in the abstraction of the main goals of program codes, let them practice Representation tasks. If you want to help them improve in the abstraction of the control flow of program codes, let them practice either Recall or Representation tasks or both of them. To help them improve in the abstraction of elementary operations in program codes, let them practice Recall tasks. On the other hand, to help *medium novices, i.e., moderate programming level novices,* improve in the abstraction of the main goals of program codes, let them practice Renaming Task, then Tracing tasks as the latter is more effective than the former. To help them improve in the abstraction of the control flow of program codes, let them practice either Tracing or Renaming or both of them. To help them improve in the abstraction of the data flow of program codes, let them practice Renaming Task, then Tracing tasks as the latter is more effective than the former. In addition, an interesting finding in the experiment was that Tracing task's effect, which is a medium cognitive category task, was comparable to the high cognitive category task's effect, i.e., Comparison and Modification. This task helped the novices improve their Block and Module abstraction levels and their related information categories. This suggests that educators can give this task to the novices at any level, i.e., introductory or advanced.

8.2 Limitations

The experiments were conducted only once at each university. A better design might be a repeated cross□sectional study to confirm the effect of tasks on the improvement of novices' mental model when used to facilitate program comprehension. Also, due to limited time assigned to the experiment, task was based on small to medium program codes, and the number of pre- and post-task questions provide was limited to three. Additionally, the use of a control group would have served as a suitable baseline to compare the effect of the intervention. Finally, the proposed model was empirically applied but not validated by experts from the program comprehension and computer science education areas.

8.3 Future Work

The research suggests interesting research trends: 1) Extending the research to validate all the classified tasks and to focus more on their effect on the higher abstraction levels, i.e., Program Level and Domain Level; 2) Extending the experiment conducted in this study via a qualitative study; 3) Use control group to additionally measure the effectiveness of the task on the improvement of the novices' mental model, and 4) Validating our proposed model by a panel of experts from the areas of program comprehension and computer science education.

Table 14: Tasks and their Effect on the Information Categories Per Students Programming Level

Programming Level	**Information Categories**			
	Goal	**Control Flow**	**Data Flow**	**Statement Goal (Elementary Operation)**
Weak	Representation	Recall/ Representation	-	Recall
Moderate	↑ Tracing Renaming	Renaming/ Tracing	↑ Tracing Renaming	-
Strong	↑ Modification Comparison	Comparison/ Modification	Comparison/ Modification	-

REFERENCES

[1] R. Brooks, "Towards a theory of the comprehension of computer programs," *International journal of man-machine studies,* vol. 18, no. 6, pp. 543-554, 1983.

[2] E. Soloway and K. Ehrlich, "Empirical Studies of Programming Knowledge," *Software Engineering, IEEE Transactions on,* vol. SE-10, no. 5, pp. 595-609, 1984.

[3] B. Shneiderman and R. Mayer, "Syntactic/semantic interactions in programmer behavior: A model and experimental results," *International Journal of Computer & Information Sciences,* vol. 8, no. 3, pp. 219-238, 1979.

[4] N. Pennington, "Comprehension strategies in programming," in *Empirical studies of programmers: second workshop*, M. O. Gary, S. Sylvia, and S. Elliot, Eds.: Ablex Publishing Corp., 1987, pp. 100-113.

[5] A. von Mayrhauser and A. M. Vans, "Program Comprehension During Software Maintenance and Evolution," *IEEE Computer,* vol. 28, no. 8, pp. 44-55, 1995.

[6] Littman, Pinto, Letovsky, and Soloway, "Mental models and software maintenance," *Journal of Systems and Software,* vol. 7, no. 4, pp. 341-355, 1987.

[7] S. Letovsky, "Cognitive processes in program comprehension," presented at the Empirical studies of programmers: 1st Workshop, Washington, D.C., United States, 1986.

[8] M. P. O'Brien, "Software comprehension–a review & research direction," 2003

[9] M.-A. Storey, "Theories, methods, and tools in program comprehension: past, present and future," in *13th International Workshop on Program Comprehension (IWPC'5)*, 2005, pp. 181-191.

[10] J. I. Maletic and H. Kagdi, "Expressiveness and effectiveness of program comprehension: Thoughts on future research directions," in *Frontiers of Software Maintenance, 2008. FoSM 2008.*, 2008, pp. 31-37.

[11] J. Siegmund, "Program Comprehension: Past, Present, and Future," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016, vol. 5, pp. 13-20.

[12] Dunsmore, A., & Roper, M. (2000). A comparative evaluation of program comprehension measures. *The Journal of Systems and Software, 52*(3), 121-129.

[13] Ko, A. J., Myers, B. A., Coblenz, M. J., & Aung, H. H. (2006). An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. *Software Engineering, IEEE Transactions on, 32*(12), 971-987.

[14] Wang, Alf Inge, & Arisholm, Erik. (2009). The effect of task order on the maintainability of object-oriented software. *Information and Software Technology, 51*(2), 293-305.

[15] J. Sillito, G. C. Murphy, and K. De Volder, "Asking and answering questions during a programming change task," *IEEE Transactions on Software Engineering,* vol. 34, no. 4, pp. 434-451, 2008.

[16] J. Starke, C. Luce, and J. Sillito, "Searching and skimming: An exploratory study," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, 2009, pp. 157-166.

[17] A. T. T. Ying and M. P. Robillard, "The Influence of the Task on Programmer Behaviour," in *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*, 2011, pp. 31-40.

[18] N. Peitek, S. Apel, A. Brechmann, C. Parnin, and J. Siegmund, "CodersMUSE: Multi-Modal Data Exploration of Program-Comprehension Experiments," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 2019, pp. 126-129.

[19] R. Francese, M. Risi, and G. Tortora, "MetricAttitude++: Enhancing Polymetric Views with Information Retrieval," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, 2017, pp. 368-371.

[20] E. A. Fontana and F. Petrillo, "Visualizing Sequences of Debugging Sessions using Swarm Debugging," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 2019, pp. 139-143.

[21] C. Peterson, P. LaBorde, and D. Dechev, "CCSpec: A Correctness Condition Specification Tool," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 2019, pp. 220-230.

[22] A. Shargabi, S. A. Aljunid, M. Annamalai, S. Mohamed Shuhidan and A. Mohd Zin, "Program comprehension levels of abstraction for novices," 2015 International Conference on Computer, Communications, and Control Technology (I4CT), Kuching, 2015, pp. 211-215. doi: 10.1109/I4CT.2015.7219568

[23] C. Izu *et al.*, "Fostering Program Comprehension in Novice Programmers-Learning Activities and Learning Trajectories," in *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, 2019, pp. 27-52.

[24] M. Begum, J. Nørbjerg, and T. Clemmensen, "Strategies of Novice Programmers," in *Proceedings of the 41st Information Systems Research Seminar in Scandinavia: Digital Adaptation, Disruption and Survival (IRIS2018)(IRIS). http://hdl. handle. net/10398/9686*, 2018.

[25] Simon and S. Snowdon, "Explaining program code: giving students the answer helps - but only just," presented at the Proceedings of the seventh international workshop on Computing education research, Providence, Rhode Island, USA, 2011. Available: https://doi.org/10.1145/2016911.2016931

[26] J.-M. Burkhardt, F. Détienne, and S. Wiedenbeck, "Object-Oriented Program Comprehension: Effect of Expertise, Task and Phase," *Empirical Software Engineering,* vol. 7, no. 2, pp. 115-156, 2002.

[27] B. Du Bois, S. Demeyer, and J. Verelst, "Does the "Refactor to Understand" reverse engineering pattern improve program comprehension?," in *Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on*, 2005, pp. 334-343.

[28] N. Pennington, "Stimulus structures and mental representations in expert comprehension of computer programs," *Cognitive psychology,* vol. 19, no. 3, pp. 295-341, 1987.

[29] A. von Mayrhauser and A. M. Vans, "From program comprehension to tool requirements for an industrial environment," in *Proceedings of the Second IEEE Workshop on Program Comprehension* 1993, pp. 78-86.

[30] C. Schulte, "Block Model: an educational model of program comprehension as a tool for a scholarly approach to teaching," presented at the Proceeding of the Fourth international Workshop on Computing Education Research, Sydney, Australia, 2008. Available: http://portal.acm.org/citation.cfm?doid=1404520.1404535

[31] A. von Mayrhauser and A. M. Vans, "Program Comprehension During Software Maintenance and Evolution," *Computer,* vol. 28, no. 8, pp. 44-55, 1995.

[32] A. Shargabi, S. A. Aljunid, M. Annamalai, S. M. Shuhidan and A. M. Zin, "Tasks that can improve novices' program comprehension," 2015 IEEE Conference on e-Learning, e-Management and e-Services (IC3e), Melaka, 2015, pp. 32-37. doi: 10.1109/IC3e.2015.7403482

[33] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer, 2012.

[34] Basili, Selby, Richard W, & Hutchens, David H. (1986). Experimentation in software engineering. *IEEE Transactions on Software Engineering,* (7), 733-743.

[35] R. Lister *et al.*, "A multi-national study of reading and tracing skills in novice programmers," presented at the Working group reports from ITiCSE on Innovation and technology in computer science education, Leeds, United Kingdom, 2004.

[36] P. Deitel and H. Deitel, *Java How to program*. Prentice Hall, 2012.

[37] Sheskin, David J. (2011). *Handbook of Parametric and Nonparametric Statistical Procedures, Fifth Edition*: CRC PRESS.

[38] Lahtinen, Essi, Ala-Mutka, Kirsti, & Järvinen, Hannu-Matti. (2005). A study of the difficulties of novice programmers. *SIGCSE Bull., 37*(3), 14-18. doi: 10.1145/1151954.1067453.

[39] Campbell, Donald Thomas, Stanley, Julian C, & Gage, Nathaniel Lees. (1963). Experimental and quasi-experimental designs for research: Houghton Mifflin Boston.

[40] Cook, Thomas D, & Campbell, Donald Thomas. (1979). *Quasi-experimentation: Design and analysis for field settings*: Rand McNally.

[41] J. Siegmund, "Framework for Measuring Program Comprehension," PhD, 2012.