INTERNATIONAL JOURNAL OF ACADEMIC RESEARCH IN PROGRESSIVE EDUCATION & DEVELOPMENT

OPEN ACCESS JOURNAL

Vol 10, Issue 3, (2021) E-ISSN: 2226-6348

Program Comprehension Technique in Teaching and Leaning: A Cognitive Perspective

Rozita Kadar[1], Putra Sumari[2], Jamal Othman[3], Syarifah Adilah Mohamed Yusoff[4]

[1,3,4]Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA Cawangan Pulau Pinang Malaysia, [2]School of Computer Sciences, Universiti Sains Malaysia Pulau Pinang, Malaysia

To Link this Article: http://dx.doi.org/10.6007/IJARPED/v10-i3/11570 DOI:10.6007/IJARPED/v10-i3/11570

Published Online: 21 *September* 2021

Abstract

In studying about programming languages, the important part is to understand the language itself. Learners need to be able to comprehend a program that is completed with syntax, semantic and program flow. Most learners especially the novices face a lot of problems when trying to learn a program. Many studies have been conducted to observe the process on how learners understand the program source code. Usually, the study of program comprehension focuses on the combination of two important characteristics: theories and tools. The theories that provide how to improve program comprehension and tools that can implement the theories. These two characteristics will change the way programmers understand the program codes. Many researchers review some of the key theories of program comprehension and discusses on how these theories are related to tools that support it. Thus, the aim of this study is to explore the evolution of the three predominant approaches of program comprehension in the aspect of cognitive theory which are: bottom-up, top-down and the integrated approach. This study also considers the important of cognitive model to make the effective learning process. Therefore, this paper can provide the intuitive environment for the process of learning especially for novice learners.

Keywords: Program Comprehension, Cognitive Theory, Cognitive Model, Domain Knowledge

Introduction

Program comprehension is “the process of taking source code and understanding it” (Deimel and Naveda 1990) or the process of using the existing knowledge to acquire new knowledge (Aljunid, Zin, & Shukur, 2012; Xia et al., 2017). The understanding on program is related to execution behaviour and relationship of variables involved in the program (Hu et al., 2018; Sasirekha and Hemalatha, 2011). The study of program comprehension can be explained as the process occurs in the learners’ mind when they understand a program (Feigenspan and Siegmund, 2012).

Source code is a more trusted source of data compared to composed documentation primarily since documentation is regularly non-existed or obsolete (Maalej et al., 2014).

However, the problems still exist if the source code is used as reference to a system. The activity in reviewing and understanding a source code is not the same as reviewing ordinary documents and many problems in program comprehension arise due to the use of textual representation as the primary source of information. In fact, programs are often in the form of a hierarchical structure, but the actual behaviour of a program cannot be reflected as it is represented in textual forms. Although many methods and tools have been proposed to represent source code, experience have shown that textual presentation is the most suitable to represent the software system (Krinke, 2004). Despite many studies carried out in finding the different strategies and techniques to overcome program comprehension problems, most researchers have yet to discuss on how to help the software maintainers to comprehend a program.

The remaining paper is divided into the following main sections: section 2 describes the methodology of the study (plan, conduct, and analysis), followed by section 3 discusses the findings to research questions. In the end, conclusions are made in Section 4.

Methodology

This section explains the methodology conducting in this study by examining, exploring and classifying the present literature according to the cognitive domain. According to Kitchenham (Kitchenham et al., 2009) conducting the literature review are classified into three levels (i) Planning, (ii) Conducting and (iii) reporting the review. We followed methodology in this paper. In figure 1, we have shown the overall process in conducting this study.

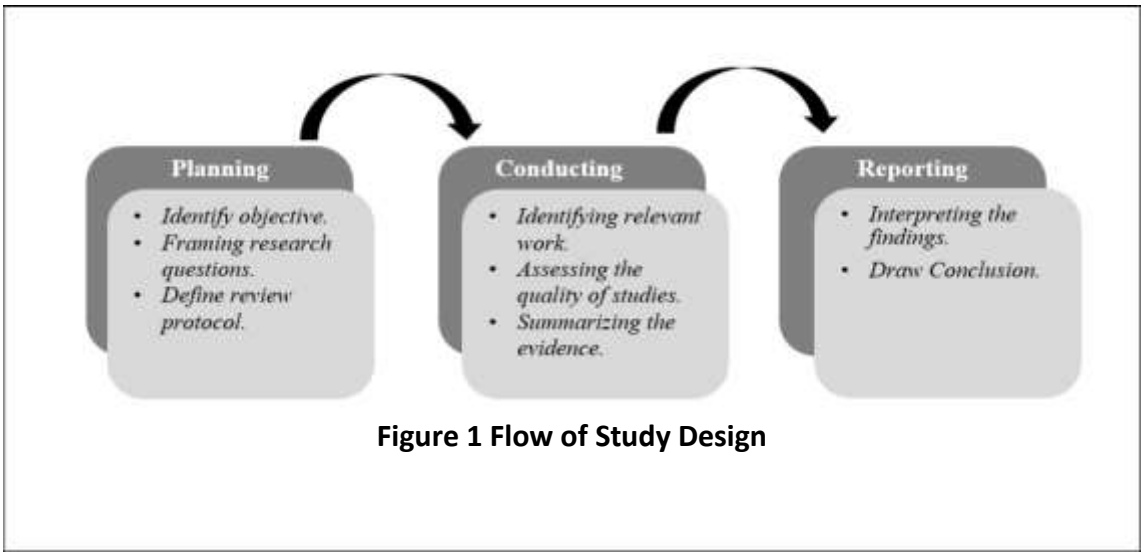

Figure 1 Flow of Study Design

Research Objective

The objective of this research is as below

1. To identify the types of knowledge, involve in program comprehension process.
2. To explore the approaches implementing in program comprehension.
3. To present the different existing models for program comprehension.

Research Question

Through this work, we answer those following three research questions (RQs) as shown in

Table 1:

Table 1 Research Questions

ID	Research Question	Motivation
RQ1	What types of knowledge are applying in program comprehension process?	To understand the different types of knowledge in cognitive domain.
RQ2	What approaches are used to improve the performance of learning a program?	To study the different types of existing approaches to improve learning process of a program.
RQ3	What are the existing cognitive models in the fields of program comprehension?	To identify the existing cognitive models of program comprehension.

Knowledge Representation in Program Comprehension Process

Frequently, most of the studies on program comprehension consider three basic elements that complement to the comprehension process. These elements are knowledge based that appear in a programmer's mind; external representation and; the assimilation process (in Figure 2). The process flow explains how developers understand a program using their existing knowledge through the assimilation process supported by external representation to obtain new knowledge.

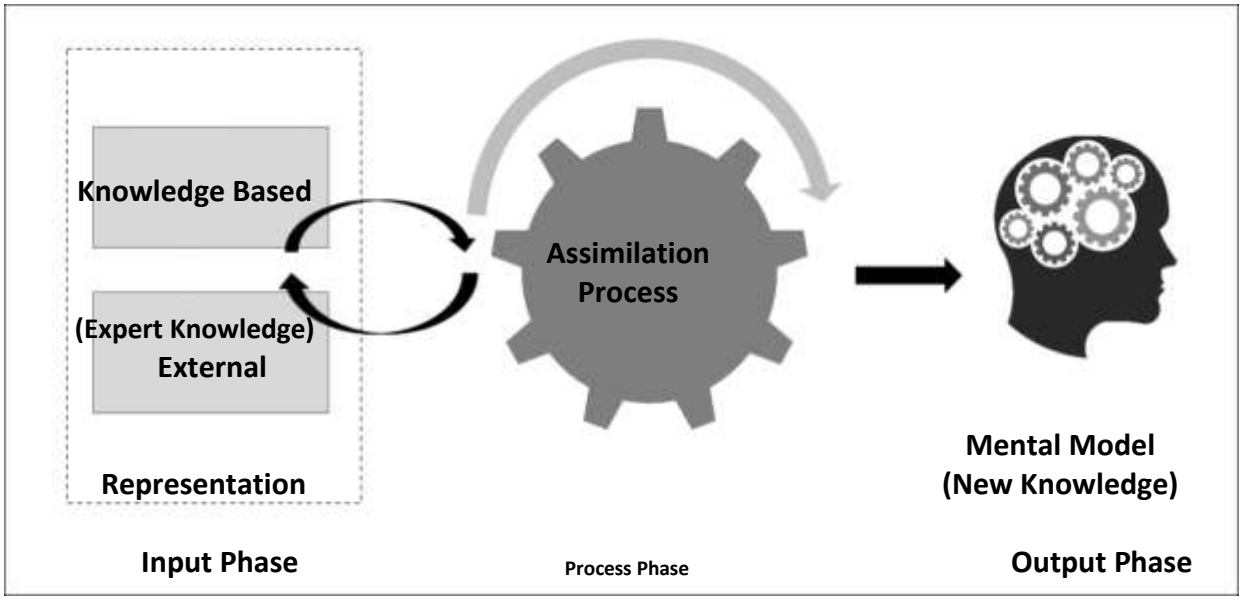

Figure 2 The Program Comprehension Process

Knowledge-based

Knowledge based is an experience or existing knowledge on a program contained by a programmer. It can determine the programmers' ability to comprehend a program. Table 2 shows the types of knowledge and researchers discovering the ideas.

Table 2 Types of Knowledge

Authors	Types of knowledge
Brooks, 1983; Carvalho, 2013; Rugaber, 2000; von Mayrhauser & Vans, 1997	Domain Knowledge
Soloway & Ehrlich, 1984; Wiedenbeck, 1986	Plans and Rules of discourse
Brooks, 1983; Rist, 1986; Weiss & Mockus, 2013	Beacons and Chunks
Gellenbeck & Cook, 1991; Shneiderman & Mayer, 1979; Soloway & Ehrlich, 1984	Syntactic and Semantic
Busjahn *et al.*, 2014; Détienne, 2002; El-sheikh *et al.*, 2013; Letovsky, 1987; Piaget, 2013)	Schemas and Abstraction

Domain knowledge consists of three domains, which are task/problem domains, intermediate domain and program domain (Brooks, 1983). During the comprehension process, the task domain is mapped to the intermediate domain and produces the program domain. Moreover, hypotheses can be constructed using domain knowledge by predicting the program with reference to the existing knowledge. Another types of knowledge is the plan and rules of discourse, which is used for developing and validating expectations, interpretations and

inferences, includes causal knowledge on information flow and the relationships among parts of a program (Soloway and Ehrlich, 1984; Wiedenbeck, 1986).

Beacons are the familiar feature in the source code serving as a cue indexed into existing knowledge to present certain structure of plans (Brooks, 1983; Rist, 1986). On top of that, beacons are utilised to predict hypotheses. Another type of knowledge is schema. According to Piaget's theory, schemas are the way of organising knowledge to become as a unit. Each knowledge is related to aspects including the object, action and abstract concepts (Piaget, 2013).

External Representations

External representations are any materials available as an aid to support programmers while comprehending a program. The materials can be represented in different ways and formats. The external support may be in a form of system documentation, source code, manual, book or expert advises as well as techniques and tools.

Assimilation Process

Assimilation is a process comprehending a program and considering incorporated and constructed with existing knowledge. In particular sign, the characteristics of programmers while comprehending a program is important since they use all their senses and capabilities to understand a program.

Program Comprehension Approach

Cognitive model is used to represent the processes involved in developing and building the programmers' mental model or acquire new knowledge from existing knowledge (Storey 2006). Developers use their existing knowledge such as programming expertise, programming language, computing environments, programming principles, architectural model, algorithm and solution approve as well as domain-specific information or problem-domain, which will after that go through the assimilation process supported by external representation. This process is continued to obtain new knowledge like functionality, architecture, algorithm implementation, control flow and data flow.

Previous studies on cognitive model provide explanations on the short-, long-,and working-memories used (Brooks 1983; von Mayrhauser and Vans 1997; Pennington 1987; Soloway and Ehrlich 1984). Other authors theorised that cognitive internal representation of knowledge is produced through the concept of frames, plans, and chunks (Minsky,1974; Rich & Waters, 1990; Soloway, 1984). Gagne (1985) also proposed cognitive strategies and believed that environment can influence the comprehension process. He stated that to stabilise the cognitive strategies, people must have certain techniques of thinking, ways of analysing problems and having approaches in solving a problem. People use cognitive strategies in thinking about the things they learnt and in solving problems. These are the ways in managing the processes of learning, remembering and thinking. Bloom (1956) discovered the ideas of learning domain called Bloom's Taxonomy and adjusted by Anderson et al. (2001). The taxonomy focuses on three domains with one of them devoted on cognitive domain that emphasises things that learners to know during learning. It involves knowledge and the ability to develop intellectual skills.

Bandura (1994) argued that people can gain new knowledge through viewing or observing. He stated the steps involved in learning process, which are attention, retention, reproduction and motivation. The first learning step proposed is to pay attention to new things. Learners have to pay full attention to grasp a new knowledge. Then, they must have the ability to store the information (retention) they obtained. This internal mental state is important as an essential part in the learning process. The next step is the reproduction where learners are able to use the knowledge they grasp and to be successful in their learning, they have to be motivated to apply the new knowledge modelled.

The next is the discussion on the three predominant approaches of program comprehensions, which are top-down, bottom-up and integrated meta-model. These models are the foundation in creating the new model of program comprehension (Meng et al., 2006; O'brien, 2003).

Top-down

Typically, top-down approach is adopted when the developers become familiar with the source code (Soloway and Ehrlich 1984). This approach is goal-oriented and hypothesis-driven contains a hierarchy of goals and plans. It is the dynamic process strategy of reconstructing knowledge to formulate hypotheses regarding the domain of the program and mapping this knowledge to the source code and use the strategic plan to implementation plan (Brooks 1983; Von Mayrhauser and Vans 1995; Storey 2005). However, the limitation of this approach is that it does not consider novices' capabilities as they are inexperienced in the domain and lack of knowledge to formulate hypotheses in the first place

Bottom-up

Bottom-up approach is introduced by Letovsky (1987) focusing on novice developers since it does not require higher level knowledge structures such as design or application-domain knowledge. The developer firstly read the code statements and then mentally chunk or group these lines of code into higher-level abstractions to form the abstract concepts supported by beacons (Letovsky 1987; Von Mayrhauser and Vans 1995). This approach is suitable for developers who are unfamiliar with the source code.

Integrated Meta-model

Von Mayrhauser & Vans (1995) introduced the integrated meta-model by integrating the top-down and bottom-up approaches. The proposed approach is based on the observations. They found that neither top-down nor bottom-up is the best approach in the assimilation process (von Mayrhauser and Vans 1993). Supported by Storey (2005), the paper mentioned that developers can choose to invoke top-down or bottom-up model as a starting point for formulating hypotheses when the code is familiar.

Program Comprehension Cognitive Models

This section discusses the existing models supporting program comprehension. The discussion is based on selected papers focusing on the strategies, approaches and the process taken to assimilate the existing knowledge to yield new knowledge. Figure 3 shows the evolution of program comprehension discussed by Schulte et al. (2010). In previous studies, most of the researchers used the cognitive model as a strategy to propose a program

comprehension model. They believe that this is the way of managing the processes of learning, remembering, and thinking.

The first model proposed by Shneiderman & Mayer (1979) uses the bottom-up approach focusing on novice users. The model involves the short-term memory and long-term memory as well as the internal semantic knowledge to develop mental model. It involves a process of chunking in which users are mentally making a chunk out of a program guided by the beacons. Pennington (1987) proposed a model with bottom-up approach guided by beacons, plans and text structure to perform chunking process. The work integrated the domain and program model to depict situation model. Burkhardt et al (2002) in their study use bottom-up approach to comprehend Object-Oriented Program compared to Shneiderman & Mayer (1979); Pennington (1987) that focuses on structured program.

Instead of cognitive models, the combination of other models were applied in the studies such as text comprehension model (Pennington 1987), constructivist model (Exton, 2002; Václav et al., 2002), vision model (Ali et al., 2011) and problem solving model (Douce, 2008). Learning Model proposed by Rajlich & Wilde (2002) interprets programmes based on constructivist theory where the developer divides program comprehension process into assimilation and adaptation. From his perspective, assimilation is the process of adding new facts to mental model, otherwise adaptation is the process of organising the existing knowledge to absorb new knowledge. Xu (2005) extends the Learning Model, namely Multi-Dimensional Model integrating the Bloom's Taxonomy, cognitive model and learning model. This study looks at the activities of assimilation and accommodation in the learning process. Although this study focuses on experts to make a hypothesis, it is also suitable for novices as it combines top-down and the bottom-up approaches.

Meng et al. (2006) introduced the Comprehension Process Model that utilises ontology and the description logic to constitute the content of mental model. The ontology based on story-drive is used to model the sources of information that describes the behaviour of a program. Store Model proposed by Douce (2008) is the heuristic model combining the elements in the working memory model and other knowledge such as strategic, semantic and plan. Frey et al (2011) worked on categorisation and separation of concern to build a mental model. Their study took an element in programmers' knowledge to understand the program of concern. The process
makes use of prediction or hypothesis using prior knowledge and verification on the prediction will update the knowledge about the concern. The classification of the models was made as shown in Table 3.

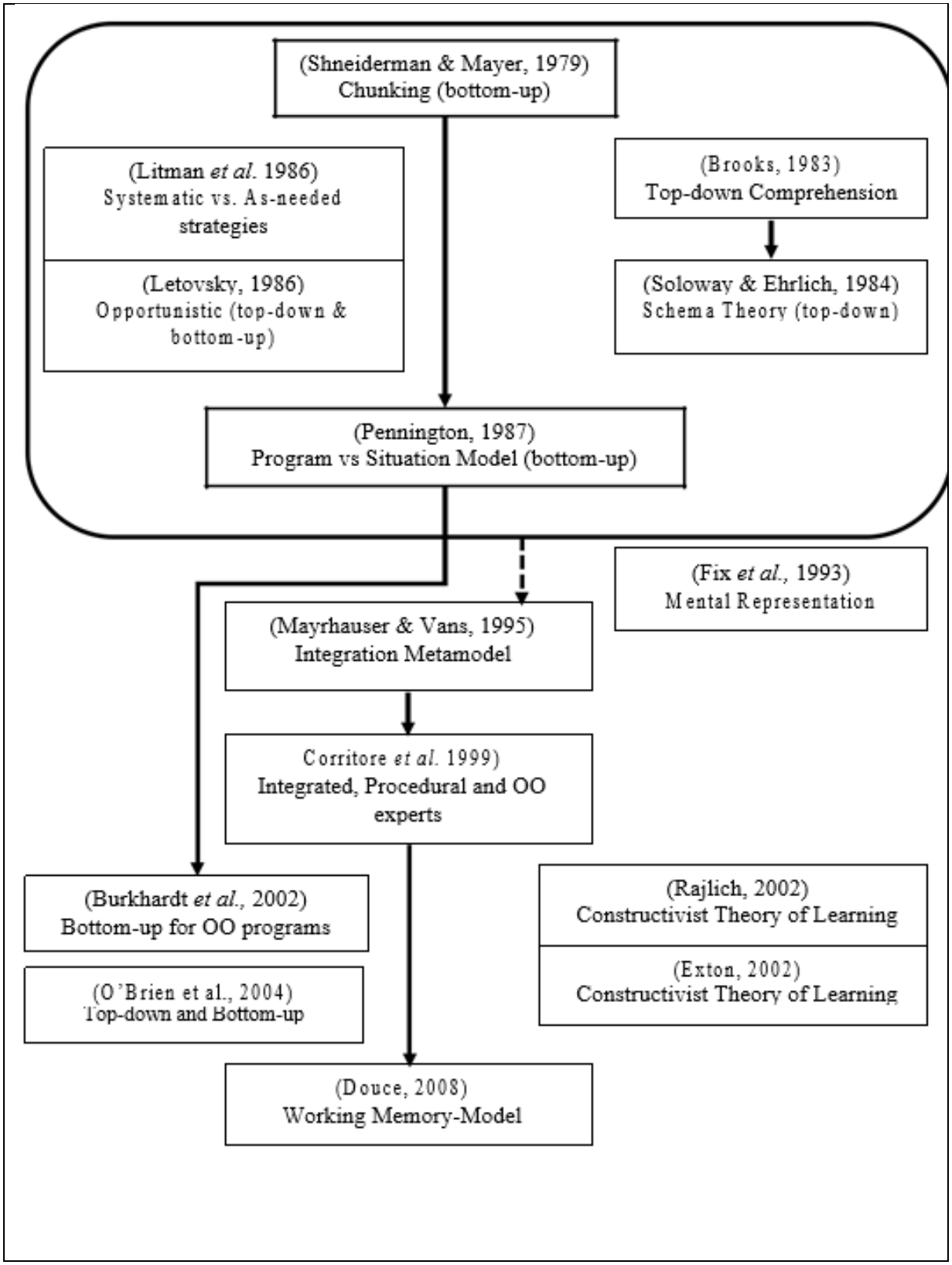

Figure 3 Evolution of Program Comprehension Model

Table 3
The Classification of Program Comprehension Model

Authors	Approaches	Comprehension Process	Types of Knowledge
Burkhardt *et al.*, 2002; Pennington, 1987; Shneiderman & Mayer, 1979	Bottom-up	Chunks	Syntactical and semantic, beacons, plans
Brooks, 1983; Vaclav *et al.*, 1994; Soloway & Ehrlich, 1984	Top-down	Hypothesis Chunks	Beacons, schema, rules of discourse, plans
Ali *et al*., 2011; Douce, 2008; Frey *et al.*, 2011; Letovsky, 1987; von Mayrhauser & Vans, 1997; Xu, 2005)	Integrated	Chunks Hypothesis	Domain expertise, domain goal, rules of discourse, plans, beacons

Conclusion and Future Work

This study is important in order to help novice learners to improve their comprehension in their learning process. Besides, this study can be seen as a form of guideline for researchers who is interested in developing tools that focus on improving comprehension. Besides, this study can contribute as an introduction in learning programming language for the novices. Future work will focus on the development of new technique in order to improve understanding in program comprehension activity. As well as to produce effective tool that can be used as an aid among novice learners when they learn a programming language; and to measure the significance of the proposed technique and its tool in improving comprehension if it is used among novices.

Corresponding Author
Rozita Kadar
Faculty of Computer and Mathematical Sciences Universiti Teknologi MARA Cawangan Pulau Pinang MALAYSIA
Email: rozita231@uitm.edu.my

References

Ali, N., Gueheneuc, Y.-G., & Antoniol, G. (2011). Requirements Traceability for Object Oriented Systems by Partitioning Source Code. *2011 18th Working Conference on Reverse Engineering*, 45–54. https://doi.org/10.1109/WCRE.2011.16

Aljunid, S. A., Zin, A. M., & Shukur, Z. (2012). A Study on the Program Comprehension and Debugging Processes of Novice Programmers. *Journal of Software Engineering*, *6*(1), 1-9

Anderson, L. W., Krathwohl, D. R., & Bloom, B. S. (2001). *A taxonomy for learning, teaching,*

and assessing: A revision of Bloom's taxonomy of educational objectives. Allyn & Bacon.

Bandura, A. (1994). *Self-efficacy*. Wiley Online Library.

Bloom, B. S. (1956). Taxonomy of Educational Objectives. Vol. 1: Cognitive domain. *New York: McKay*.

Brooks, R. (1983). Towards A Theory of the Comprehension of Computer Programs. *International Journal of Man-Machine Studies*, *18*(6), 543–554.

Burkhardt, J.-M., Détienne, F., & Wiedenbeck, S. (2002). Object-Oriented Program Comprehension: Effect of Expertise, Task and Phase. *Empirical Software Engineering*, *7*(2), 115–156.

Busjahn, T., Schulte, C., & Kropp, E. (2014). Developing Coding Schemes for Program Comprehension using Eye Movements. *Proceedings 25th Annual Workshop of the Psychology of Programming Interest Group*, 111–122.

Deimel, L. E., & Naveda, J. F. (1990). Reading Computer Programs: Instructor's Guide to Exercises. In *Educational Materials CMU/SEI-90-EM-3*. DTIC Document.

Douce, C. (2008). The Stores Model of Code Cognition. *Programmer Psychology Interest Group*, 1–10. Retrieved from http://www.ppig.org/papers/20th-douce.pdf

El-Sheikh, E., Reichherzer, T., White, L., Wilde, N., Coffey, J., Bagui, S. & Baskin, A. (2013). Towards Enhanced Program Comprehension for Service Oriented Architecture (SOA) Systems. *Journal of Software Engineering and Applications*, *6*(09), 435.

Exton, C. (2002). Constructivism and Program Comprehension Strategies. *Program Comprehension, 2002. Proceedings. 10th International Workshop On*, 281–284.

Feigenspan, J., & Siegmund, N. (2012). Supporting Comprehension Experiments With Human Subjects. *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, (6), 244–246. https://doi.org/10.1109/ICPC.2012.6240494

Frey, T., Gelhausen, M., & Saake, G. (2011). Categorization of Concerns: A Categorical Program Comprehension Model. *Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools*, 73–82. ACM.

Gellenbeck, E. M., & Cook, C. R. (1991). An Investigation of Procedure and Variable Names as Beacons During Program Comprehension. *Proceedings of the Fourth Workshop on Empirical Studies of Programmers*, 65–79.

Hu, X., Li, G., Xia, X., Lo, D., & Jin, Z. (2018). Deep Code Comment Generation. *Proceedings of the 26th Conference on Program Comprehension*, 200–210. ACM.

Krinke, J. (2004). Visualization of Program Dependence and Slices. *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, 168–177. https://doi.org/10.1109/ICSM.2004.1357801

Letovsky, S. (1987). Cognitive Processes in Program Comprehension. *Journal of Systems and Software*, *7*(4), 325–339.

Maalej, W., Tiarks, R., Roehm, T., & Koschke, R. (2014). On the Comprehension of Program Comprehension. *ACM Transactions on Software Engineering and Methodology*, *23*(4), 1–37. https://doi.org/10.1145/2622669

Meng, Wen, Rilling, J., Zhang, Y., Witte, R., Mudur, S., & Charland, P. (2006). A Context-Driven Software Comprehension Process Model. *2006 Second International IEEE Workshop on Software Evolvability (SE'06)*, 50–57. https://doi.org/10.1109/Software-Evolvability.2006.1

O'brien, M. P. (2003). Software Comprehension–A Review and Research Direction. *Department of Computer Science & Information Systems University of Limerick, Ireland, Technical Report*.

Pennington, N. (1987). Comprehension Strategies in Programming. *Empirical Studies of Programmers: Second Workshop*, 100–113. Ablex Publishing Corp.

Piaget, J. (2013). *The construction of reality in the child* (Vol. 82). Routledge.

Rajlich, Vaclav, Doran, J., & Gudla, R. T. S. (1994). Layered Explanations of Software: A Methodology for Program Comprehension. *Program Comprehension, 1994. Proceedings., IEEE Third Workshop On*, 46–52. IEEE.

Rajlich, V., & Wilde, N. (2002, June). The Role of Concepts in Program Comprehension. In *Proceedings 10th International Workshop on Program Comprehension* (pp. 271-278). IEEE.

Rist, R. S. (1986). Plans in Programming: Definition, Demonstration, and Development. *First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*, 28–47.

Rugaber, S. (2000). The Use of Domain Knowledge in Program Understanding. *Annals of Software Engineering*, *9*(1–2), 143–192.

Sasirekha, N., & Hemalatha, M. (2011). Program Slicing Techniques and its Applications. *International Journal of Software Engineering & Applications*, *2*(3), 50–64.

Shneiderman, B., & Mayer, R. (1979). Syntactic/Semantic Interactions in Programmer Behavior: A model and experimental results. *International Journal of Computer & Information Sciences*, *8*(3), 219–238.

Soloway, E., & Ehrlich, K. (1984). Empirical Studies of Programming Knowledge. *Software Engineering, IEEE Transactions On*, (5), 595–609.

Storey, M.-A. (2005). Theories, Methods and Tools in Program Comprehension: Past, Present and Future. *Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop On*, 181–191. IEEE.

Storey, M.-A. (2006). Theories, Tools and Research Methods in Program Comprehension: Past, Present and Future. *Software Quality Journal*, *14*(3), 187–208.
https://doi.org/10.1007/s11219-006-9216-4

Von Mayrhauser, A., & Vans, A. M. (1995). Program Understanding: Models and Experiments. *Advances in Computers*, *40*, 1–38.

Weiss, D. M., & Mockus, A. (2013). The chunking pattern. *Data Analysis Patterns in Software Engineering (DAPSE), 2013 1st International Workshop On*, 35–37. IEEE.

Wiedenbeck, S. (1986). Beacons in Computer Program Comprehension. *International Journal of Man-Machine Studies*, *25*(6), 697–709.

Xia, X., Bao, L., Lo, D., Xing, Z., Hassan, A. E., & Li, S. (2017). Measuring Program Comprehension: A Large-Scale Field Study with Professionals. *IEEE Transactions on Software Engineering*, *44*(10), 951–976.

Xu, S. (2005). A Cognitive Model for Program Comprehension. *Third ACIS Int'l Conference on Software Engineering Research, Management and Applications (SERA'05)*, 392–398. https://doi.org/10.1109/SERA.2005.2