Systematic Literature Review of Empirical Studies on Mental Representations of Programs☆,☆☆

Leah Bidlake[a,*], Eric Aubanel[a], Daniel Voyer[b]

[a]*Faculty of Computer Science, University of New Brunswick, 550 Windsor St, Fredericton, NB, E3B 5A3, Canada*
[b]*Department of Psychology, University of New Brunswick, Canada*

Abstract

Programmers are frequently tasked with modifying, enhancing, and extending applications. To perform these tasks, programmers must understand existing code by forming mental representations. Empirical research is required to determine the mental representations constructed during program comprehension to inform the development of programming languages, instructional practices, and tools. To make recommendations for future work a systematic literature review was conducted that summarizes the empirical research on mental representations formed during program comprehension, how the methods have changed over time, and the contributions of the research.

The data items included in the systematic review are empirical studies of programmers that investigated the comprehension and internal representation of code written in a formal programming language. The eligibility criteria used in the review are meant to extract studies with a focus on knowledge representation as opposed to knowledge utilization.

The results revealed a lack of incremental research and a dramatic decline in the research meaning that newly developed or popularized languages and paradigms have not been a part of the research reviewed. Accordingly, we argue that there needs to be a resurgence of empirical research on the psychology of programming to inform the design of tools and languages, especially in new and emerging paradigms.

Keywords: Mental Representations, Program Comprehension, Systematic Literature Review

1. Introduction

Programmers are frequently tasked with modifying, enhancing, and extending applications. To perform these tasks, programmers must first understand the existing code. During the comprehension process, programmers form mental representations of the code they are working with (Détienne, 2001). Empirical research is required to determine how these mental representations are created and the form that they take as a starting point to developing programming languages and tools that fit with the underlying representations. Such an approach would assist programmers in the comprehension process and promote the formation of accurate mental representations.

Research in program comprehension encompasses both the study of the cognitive processes used by programmers to understand code and how programming languages and tools support these cognitive processes (Storey, 2006). The cognitive component of program comprehension that is of interest here is the abstract mental representations that are formed during program comprehension. These mental representations, often referred to as mental models, are founded in the theories of text comprehension (Pennington, 1987a).

There have been a variety of differing approaches to mental models including how they are defined and inferred (Cañas and Antolí, 1998). Cañas and Antolí suggested that the reason for this is because researchers from different disciplines study mental models using different tasks and are sometimes interested in different aspects of the representations. The unifying definition for a mental model, proposed by Cañas and Antolí (1998), is a dynamic representation formed in working memory as a result of using knowledge from long term memory and the environment. Mental model representations are described by Cañas and Antolí (1998) as both a process and the result of a simulation process elicited by a task.

The mental model approach to program comprehension is based on the propositional or text-based model and the situation model that were first developed to describe text comprehension (Détienne, 2001). The program model, formed by programmers when applying structural knowledge to the code resulting in a surface level representation, corresponds to the propositional or text-based model formed during text comprehension. The situation model, developed to describe the abstract representation of text, corresponds to the domain model formed during program comprehension when domain knowledge is used to form an understanding of the real world situation represented by the program. The mental model approach to program comprehension involves the construction of both the program model and

☆Declarations of interest: none
☆☆This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors
*Corresponding author
Email addresses: `leah.bidlake@unb.ca` (Leah Bidlake), `aubanel@unb.ca` (Eric Aubanel), `voyer@unb.ca` (Daniel Voyer)

the domain model (Détienne, 2001).

Empirical research on program comprehension has a direct impact on the development of programming languages and tools. Boshernitsan et al. (2007) developed iXj, a tool that uses a visual language to allow programmers to specify and execute code changes. The design of iXj was guided by the Cognitive Dimensions framework developed by Green (1989) to provide programmers with visual representations that reflect their own mental model of the source code. Tubaishat (2001) developed a theoretical model, Conceptual Model for Software Fault Localization (CMSFL), from empirical research on programming knowledge and plans. The CMSFL model was then used as the basis for developing the BUG-DOCTOR, an Automated Assistant Fault Localization (AASFL) tool that assists programmers with software fault localization. Arab (1992) developed a tool for formatting and documenting Pascal programs to assist programmers to write more readable and easier to understand programs. The development of this tool was influenced by empirical research that identified formatting and documenting as important factors in program comprehension.

Other common approaches to empirical research on programming languages and tools have been usability and comparative studies. Usability studies examine the human use of programming languages, whereas comparative studies compare the features of different programming languages. The usability studies reviewed by Hornbæk (2006) employed a variety of measures and definitions of usability, and often compared previous or competing versions. Comparative studies, including those performed by Prechelt (2000) and Nanz and Furia (2015), analyzed performance of programs written in programming languages from procedural, functional, and scripting paradigms, and also compared languages within these paradigms. Performance in these studies was measured using quantitative analysis of the program features such as lines of code, runtime, memory usage, and reliability. Programmer effort was also analyzed by Prechelt (2000) who measured the time required to write a program. Comparative studies on parallel programming languages have taken a similar approach in comparing solutions to problem sets or algorithms written in different parallel programming languages (Feo, 2015; Chamberlain et al., 2000). The study conducted by Feo (2015) also compared qualitative measures of how the programmers who wrote the solutions felt about how easy or difficult the program was to write, and what they felt was good or bad about the programming language.

The usability and comparative studies mentioned here are comparing programming languages that are already widely used. Studying a language when it is already widely used offers limited advances unless one is willing to modify the language as a result of the empirical observations. These studies did not take into consideration the user and their ability to understand and provide maintenance for programs written in different programming languages. Comparative studies are also limited in that they only give the user choice between existing languages; they do not direct the development of new languages that may require a complete departure from the current approaches to provide the user with languages and tools that align with their cognitive processes. The shift from comparative studies to research on program comprehension, especially in parallel programming where there is a significant lack of theory (Mattson and Wrinn, 2008), is necessary to inform the development of programming languages, instructional practices, and tools.

The focus on program comprehension is of particular interest as many of the tasks performed by programmers, such as maintaining and modifying existing applications, require the understanding of code (Corritore and Wiedenbeck, 1999). Empirical research is required to develop an understanding of the comprehension strategies used by programmers. With the growing number of programming languages and tools under development with the goals of increasing productivity and ease of learning and use, there is a great need for this empirical research.

Before conducting more empirical research on mental representations, we need to know the current state of affairs in this area. Knowledge of past empirical research can assist in developing methods that can more accurately target and capture the mental representations under study. By discovering what research has already been done, we can also learn what gaps remain.

Accordingly, the purpose of the literature review conducted here is to provide an analysis of the research that has been done to advance the body of knowledge related to the psychology of programming. The present review summarizes the empirical studies that have contributed to the understanding of the comprehension strategies used by programmers to understand code, and the mental representations created in this process. Empirical studies related to problem solving, program design and development, debugging skills, programmer performance, and learning programming languages that have no bearing on mental representations are not included in this review. The research questions addressed by the present review are as follows:

1. To date, what empirical research has examined mental representations during program comprehension?
2. How have the methods used in the studies changed over time?
3. What are the contributions that have resulted from empirical research on program comprehension?

2. Current Study

To answer the research questions posed in this study, we conducted a systematic review. Systematic reviews are performed to collect literature relevant to specific research questions by using a well defined and documented search strategy and specific inclusion and exclusion criteria (Budgen and Brereton, 2006). The systematic review included databases from each of the relevant fields: psychology and computer science. All available publication types (e.g., theses, journal articles, conference papers, books) were included in the systematic review because the research fields involved in this study use some publication types more frequently than others. For example, computer science publications are more often conference papers whereas psychology publications tend to be journal articles. The inclusion of theses makes it more likely that unpublished work will be considered. To understand how research on program comprehension has developed over time, the review was not limited to a particular time period. The abstracts or full texts of documents extracted by the database searches were screened using specific inclusion and exclusion criteria. The eligibility criteria were used to ensure that the documents were relevant to the research questions presented earlier, which are centred around empirical studies on program comprehension.

The systematic review also included an analysis using the full text of empirical studies that met the eligibility criteria. The analysis consisted of identifying the year of the study and the tasks used to stimulate the comprehension process, and summarizing the methods used in the study and the findings. The summaries were then used to create categories that reflected the most prevalent methods and findings of the research included in the present review. Categorizing the methods and findings assisted us in answering the research questions presented earlier by allowing us to make observations about how the research on mental representations of programs has changed over time and the resulting contributions.

3. Method

This section provides a description of the systematic literature review that was conducted in accordance with PRISMA guidelines (Moher et al., 2009) and was supported using snowballing, where the reference list of data items are used to identify additional data items (Wohlin, 2014). The information sources, search process, and screening process used in this literature review are outlined in this section.

3.1. Information Sources

Five databases were included in the search process: Computer Source Index, ERIC, IEEE Xplore Digital Library, PsycINFO, and Scopus. Records in the Computer Source Index (formerly Computer Science Index) database are related to the current trends and advances in computer science. The ERIC database records are primarily related to the field of education, with publication dates from 1966 to present. The IEEE Xplore Digital Library database contains scientific and technical content published by the Institute of Electrical and Electronics Engineers (IEEE) and its publishing partners, from the fields of electrical engineering, computer science, and electronics. The publication dates range from 1872 to present. The records in the PsycINFO database refer to literature from the behavioural sciences and mental health fields of study, with publication dates ranging from 1887 to present. The Scopus database contains records from the sciences, health sciences, and social sciences, with publication dates ranging from 1966 to present.

The keyword, title, and abstract information were used to perform the database searches. The search string found in Appendix A was developed using the main components of the search: mental representations, program comprehension, and programmers. Synonyms of these components were then identified and all their combinations were searched according to the requirements of the specific database. For example, the search term for mental representation was comprised of the keyword 'schema*', and the keywords 'cognit*', 'mental*', 'knowledge', 'program*', and 'situation*' combined with 'model*', 'represent*', 'plan*', 'structur*', 'map*', 'chunk*', and 'slic*' (the wildcard symbol * is used for identifying all words starting with slic, e.g. slicing). The combined words could be at most separated by two words. The search string for the IEEE Xplore Digital Library had to be modified slightly since it was the only database that imposed a limit on the number of wildcard symbols and keywords. Because of the multidisciplinary nature of the Scopus database, its search was limited by subject area to psychology, computer science, and engineering. The date of the last search of all databases was November 8, 2018. There were no restrictions put on the publication dates or publication types (e.g., theses, journal articles, conference papers, books) when performing the database searches.

Landman et al. (2017) found that when using the IEEE Xplore database the number of results were reduced when adding an OR to their search queries. Given their concern regarding inconsistencies, multiple search queries were tested using the IEEE Xplore database. When additional OR statements were added to the search query used in the current review, the number of results increased as expected.

A request for unpublished work of relevance was sent to the Psychology of Programming Interest Group (PPIG) discussion group. As a results of this request, we received four unpublished papers.

Internet search engines were not used as an information source for the present review due to their unsystematic nature and lack of quality control. Google and Google Scholar both use personalization filters that affect the results that are returned by searches (Pariser, 2011). The use of filters by search engines creates unpredictable and inconsistent searches. The lack of standardization could create bias since searches are filtered based on what the search engine determines the user wants resulting in an incomplete retrieval of data. Internet search engines cannot exclude results that are from predatory publishers.

3.2. Search Strategy

The search results for each of the databases were as follows: Computer Source returned 65 records, ERIC returned 50 records, IEEE Xplore Digital Library returned 280 records, PsycINFO returned 92 records, and Scopus returned 879 records. In total, 1,366 data items were retrieved through database searches. There were 280 duplicate data items removed, leaving 1,158 unique data items as indicated in Figure 1.

3.3. Eligibility Criteria

A preliminary screening was performed on the 1,158 unique data items that were extracted from the five database searches. The data items were screened by reading their titles and abstracts. In cases where it was not evident if the criteria were met from the abstract, then the full text was accessed and reviewed.

Data items had to be empirical studies to be considered for inclusion during the preliminary screening. These studies also

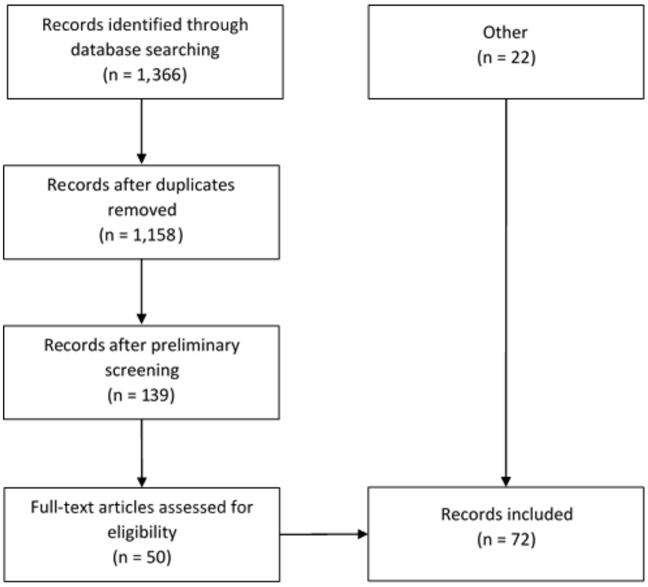

Figure 1: PRISMA Flow Diagram.

had to have participants that were programmers, meaning that they had prior knowledge of the programming language used in the study. The study had to use coded programs or code fragments that were written in a formal programming language. The flow diagram outlining the results of each screening stage is found in Figure 1. The exclusion criteria used in the preliminary screening were such that programming aptitude studies, usability studies, and studies that involved the teaching and learning of a programming language (e.g., teaching methods, educational software) were excluded, reducing the data items from 1,158 to 139 (see Figure 1). In cases where an author republished their study using the same data and analysis, only the most recent publication was kept. The full text of the documents were then analyzed to determine if they met the eligibility criteria and 50 documents were included as a result. From the use of snowballing and the unpublished work received from PPIG, 22 additional data items were included as indicated in Figure 1.

The data items that remained after the preliminary screening were subject to further scrutiny by examining the goals and methods of the studies. The elimination criteria for this stage were structured with the goal to exclude studies on the problem solving process of programming, analytic and predictive techniques for programmer performance, program design and implementation, and debugging strategies and skills. However, studies that used debugging as a task to stimulate the comprehension process in order to study programmers' mental representations were included. For example, the study conducted by Stone et al. (1990) was eliminated because they investigated the debugging skills of programmers and how these skills can be improved. The study by Petre and Blackwell (1999) was excluded because they investigated the visual imagery produced by programmers during the software design phase. Their study did not include the writing or understanding of code. The study by Kamma and Jalote (2013) was eliminated as it examined programmer productivity. Another data item eliminated from the review was a study conducted by Yeh (2014) that did not involve programming or code and investigated the cognitive processes used by participants when solving a software design problem. The criteria for inclusion at this stage required that studies had contributed to the understanding of program comprehension and mental representations.

As a result of the screening process, the data items included in the present review are empirical studies of programmers that investigated their comprehension and internal representation of code written in a formal programming language. The eligibility criteria used here are meant to extract studies with a focus on knowledge representation as opposed to knowledge utilization (Atwood and Ramsey, 1978).

4. Results

The data extracted from the 72 documents identified by the systematic review are provided in Table B.1. The items in the table are listed in chronological order to allow the reader to follow the development of the literature.

4.1. Research Timeline

Empirical studies used to develop and validate theories on program comprehension first started to emerge in the 1970's (Figure 2), with the earliest study found by the systematic review conducted in 1976. The timeline depicted in Figure 2 shows the growth and subsequent decline in the number of empirical studies that have been conducted on program comprehension. During the 1970's and 1980's, 20 studies were conducted. The number of studies grew to 40 in the 1990's when it peaked. In more recent years the work in this area has dropped off almost completely with only 12 studies conducted since 2000 (Figure 2). Throughout this timeline, various tasks have been used to stimulate the comprehension process that results in the formation of mental representations. When examining the Task column in Table B.1, it appears that, in 24 of the studies, more than one task was assigned to the participants. The most commonly assigned task was to study a program; this task was used in 51 of the studies. The remaining tasks were considerably less used, these consisted of classifying, debugging, writing documentation, enhancing, hand execution, maintenance, modification, recopying, reusing, and writing or reconstructing code.

4.2. Research Methods

To determine how the methods used in the studies included in the present review have changed over time, the method of each study has been categorized in Table B.1 for analysis. Through inspection of the Method column in Table B.1, it is apparent that a variety of techniques and measures have been used, both independently and in various combinations, to determine the mental representations formed by participants during the comprehension process. The most common technique was the use of comprehension questions (25 studies). Comprehension questions were also frequently used in addition to other measures

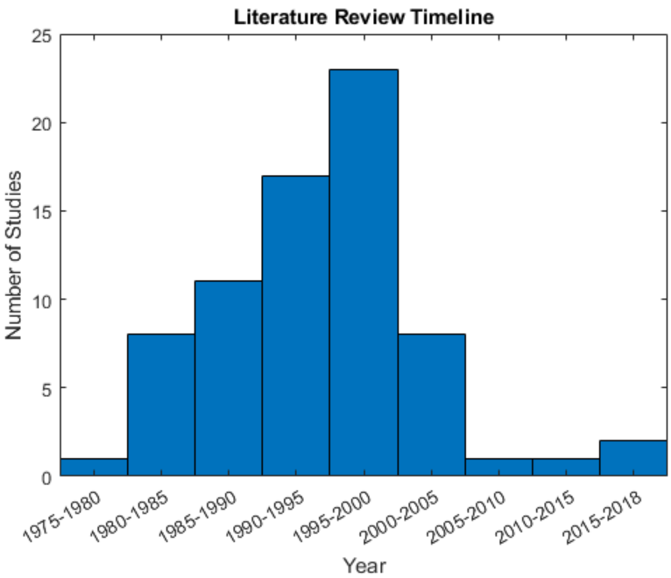

Figure 2: Frequency of empirical studies on program comprehension.

to determine the mental representations formed by participants. Each study developed its own set of comprehension questions in order to measure the strength of different mental representations that may have been formed by the participants. Although some studies tested for the same types of representations (program and domain models studied by Pennington (1987b), Corritore and Wiedenbeck (1991, 1999), Burkhardt et al. (1997), Burkhardt et al. (2002), Wiedenbeck and Ramalingam (1999), Wiedenbeck et al. (1999), Mosemann and Wiedenbeck (2001)) or knowledge structures (i.e., data flow, control flow, state, and function studied by Pennington (1987b), Teasley (1994), Shaft and Vessey (1995), Snyder (1995), Ramalingam and Wiedenbeck (1997), Khazaei and Jackson (2002)) the questions used varied between these studies. The use of verbal protocol analysis while participants performed a task and recall of code were also common techniques for measuring program comprehension (23 and 19 studies respectively). Recognition of code was used in nine studies, summarizing or describing the function of the code was used in five studies, and fill in the blank and sorting were each used three times, whereas all other techniques were each used only once. The recall of code required participants to reproduce the program used in the study either verbatim or a functionally equivalent version. The recognition of code required participants to determine if a given code fragment was from the program used in the study.

Comprehension questions form the method that has been used most consistently throughout the timeline of the present review, whereas other techniques have been introduced and in some cases, discontinued at different points along the timeline (Figure 2). The use of methods depicted in Figure 2 shows that recall was used more frequently in earlier studies but dropped off around the time that verbal protocol analysis was introduced. The use of verbal protocol analysis first appeared in a study by Letovsky (1987) and was popular until it was last used by Vans et al. (1999) and did not reappear until a study by Nosál and Porubän (2015) (Figure 3). Some of the more recent studies in the present review have introduced novel techniques for analyzing mental representations formed by programmers such as software that performed screen capture of their actions and monitoring of documents they accessed (Corritore and Wiedenbeck, 2001) and analysis of eye movement data (Fan, 2010).

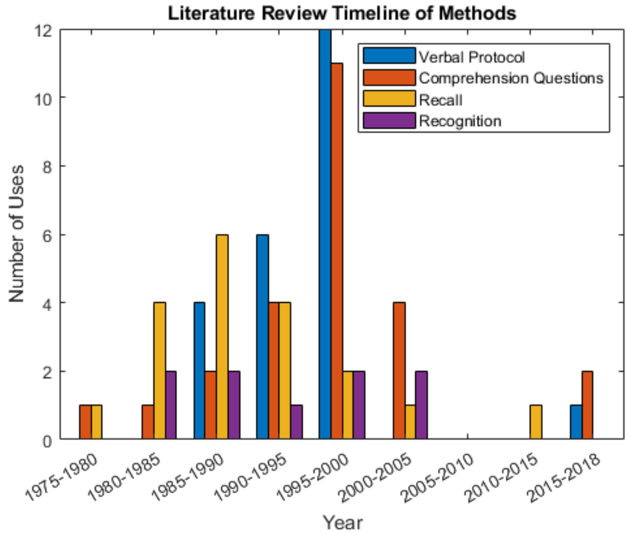

Figure 3: Frequency of most common methods used for determining the mental representations formed during the comprehension process.

The data collected in the present review and found in the Method column of Table B.1 indicate that the types of programmers compared in the studies have changed over time. From the first study in 1976 until 1990, the only types of programmers that were compared in the studies included in the present review were programmers of varying levels of expertise (Shneiderman, 1976; Adelson, 1981; McKeithen et al., 1981; Ehrlich and Soloway, 1984; Soloway and Ehrlich, 1984; Adelson, 1984; Barfield, 1986; Schmidt, 1986; Bateson et al., 1987; Boehm-Davis et al., 1987; Vessey, 1987; Vihmalo and Vihmalo, 1988; Davies, 1990b; Guerin and Matthews, 1990). Programmers were categorized as expert, intermediate, or novice to indicate their expertise in the programming language used in the study or their expertise in the domain relevant to the program. These studies used different definitions and measures to categorize participants' level of expertise. The first occurrence in the data of comparing mental representations formed by programmers trained in different programming paradigms or languages was in 1990. Since that time, comparing programmers with different backgrounds has become a more common type of comparison having been used in studies conducted by Robertson and Yu (1990), Green and Navarro (1995), Corritore and Wiedenbeck (1999), Wiedenbeck and Ramalingam (1999), Wiedenbeck et al. (1999), Corritore and Wiedenbeck (2001), Navarro-Prieto and Cañas (2001), and Khazaei and Jackson (2002). The most common comparison of programming paradigms was between procedural programming and object oriented programming.

4.3. Research Contributions

To analyze the contributions that have resulted from the research included in the present review, the findings of each study have been categorized in Table B.1. Examination of the Findings column in Table B.1 reveals considerable variation in the description of the representations formed by programmers and the strategies used during the program comprehension process. The studies conducted by Shneiderman (1976), McKeithen et al. (1981), Ehrlich and Soloway (1984), Mynatt (1984), Barfield (1986, 1997), Guerin and Matthews (1990), Furman (1998), and Fan (2010) found that programmers used chunking to develop a mental representation of the code. Chunking involves the grouping of lines of code together during the comprehension process. The strategy used by programmers when chunking differed between studies. Guerin and Matthews found that programmers chunked by identifying functions in the code; Barfield concluded that programmers grouped sequential lines of code that fit logically together; Furman found that programmers used the visual structure of the code to form chunks; and Fan determined that programmers used beacons to recognize code chunks.

Another group of studies included in the present review found that programmers formed mental representations at varying levels of abstraction. Studies performed by Pennington (1987b), Bergantz and Hassell (1991), Burkhardt et al. (1997), Burkhardt et al. (2002), Ramalingam and Wiedenbeck (1997), Corritore and Wiedenbeck (1999), Wiedenbeck and Ramalingam (1999), Wiedenbeck et al. (1999), Mosemann and Wiedenbeck (2001), and Parkin (2004) support the two model theory that programmers form low level program models and high level situation or domain models during the comprehension process. Studies conducted by von Mayrhauser and Vans (1993, 1994, 1995, 1996, 1998), Vans (1996), von Mayrhauser et al. (1997), and Vans et al. (1999) found that programmers formed mental models at three levels of abstraction and switched between them during the comprehension process, with the program model as the lowest level, the situation model as the intermediate level, and the domain model as the highest level of abstraction.

The knowledge structures and representations that compose the representations at different levels of abstraction were also investigated by a number of studies. Pennington (1987b) found that mental models were developed by programmers using a bottom-up approach (concrete to abstract) since control flow representations were used initially in the comprehension process to form program models whereas data flow and functional representations were used later to form situation models. Shaft and Vessey (1995) determined that the direction in which the representations were developed depended on the expertise of the programmer in the application domain; in an unfamiliar domain programmers developed representations in a bottom-up direction (data flow, control flow, state, and function), but in familiar domains programmers developed representations in the opposite direction (top-down). Contrary to Pennington's findings, Bergantz and Hassell (1991) concluded that control flow representations did not influence the comprehension process and that the representations used by programmers differed depending on their level of expertise: less experienced programmers developed more data flow relationships whereas more experienced programmers developed more function relationships in their mental representations. Teasley (1994) concluded that different types of knowledge structures are all acquired at a similar rate and there was no strong evidence to indicate that programmers use a bottom-up approach. Navarro-Prieto and Cañas (2001) compared programmers with backgrounds in different programming paradigms and found that procedural programmers had better developed control flow representations than data flow representations whereas visual programmers developed both representations equally well. Khazaei and Jackson (2002) compared the representations formed by programmers who had experience in both event driven and object oriented paradigms when understanding programs written in the different paradigms and found that programmers formed stronger control flow, data flow, and functional models when understanding event driven programs compared to object oriented programs. Snyder (1995) determined that the representations formed by programmers were dependent on the task they were assigned, and that modification tasks required the programmer to develop relationships between four representations: data flow, control flow, state, and function.

Contributions made by research on mental representations formed by programmers during program comprehension have been important in the development of tools and languages that are more intuitive and align with programmers' internal representations. The lack of research on mental representations of parallel programmers reinforces the need to return to this research approach to develop tools and languages for parallel programmers instead of relying on usability and comparative studies. Research that analyses the usability of tools and languages and comparative studies is not informed by theories of program comprehension or mental representations of programmers and is unable to provide insight into how programmers internalize and represent code.

5. Discussion

The purpose of the present study was to determine the extent of empirical research that examined mental representations formed during program comprehension to date, how the methods used in the research have changed over time, and the contributions that have resulted from the research.

To determine the extent of empirical research that has examined mental representations formed during program comprehension to date, a systematic review was performed and summarized in Table B.1. The timeline (Figure 2) that resulted from the systematic review indicates that recently there has been a dramatic decline in research on program comprehension. The declining number of studies focusing on program comprehension and mental representations of programmers in recent years is possibly due to a change in focus. For example, there were a number of studies conducted in recent years that focused on strategies and tools for teaching programming (Oliveira Aureliano, 2013; Dillon, 2013; Lane, 2005), learning second or subsequent programming languages (Scholtz and Wiedenbeck, 1990), and designing programs (Yeh, 2014; Basili and Reiter, 1981) that did not meet the criteria for the present review. For example, Whalley and Kasto (2014) conducted a study that examined the progression of learning and the development of cognitive structures during the learning process. Another study that was eliminated measured the amount of programmer effort required to write programs using different parallel programming models (Hochstein et al., 2008). Cañas and Antolí (1998) stated that research of mental representations was blocked due to lack of agreement in definition and methodology. The results of the current study found in Table B.1 support this conjecture by demonstrating the lack of agreement on the tasks used to elicit the mental representations and the methodology utilized to measure them. Other reasons for the observed decline may include a movement towards usability and comparative studies where there may be more funding particularly from companies wanting to demonstrate the usability of their programming languages and tools. Usability and comparative studies tend to have more concrete measures such as lines of code, runtime, and memory usage that allow a quantitative analysis of the program features providing more conclusive results.

By examining the Task and Method columns in Table B.1, we were able to determine how the studies included in the present review have changed over time. For a period of time, empirical studies on program comprehension and mental representations were performed on varying levels of programmers using a variety of programming languages. The tasks and methods used to assess and analyze program comprehension and mental representations of the participants were not consistent between studies. While Vessey (1987) criticized the use of debugging as a task to stimulate program comprehension, it has been used in studies throughout the timeline of the present review by Weiser (1981), Weiser and Ledgard (1982), Adelson (1984), Gilmore and Green (1988), Davies (1990a), Romero and Du Boulay (2004), and Fan (2010). The research performed by Wiedenbeck et al. (1993) compared expert and novice programmers who were given the task of studying code for understanding and found that studying code was an unnatural task for the experts who commented that normally they would have a concrete objective, such as debugging or predicting the effects of modifications, in mind when reading code. Wiedenbeck speculated that depending on the task, different information may be extracted during program comprehension. Wiedenbeck's observation underlines the importance of ensuring the task is appropriate for the participants in the study. In some cases researchers also expressed concerns over the validity of methods used to measure comprehension including comprehension questions (Shaft and Vessey, 1995) and recall (Guerin and Matthews, 1990) that are used in a number of other studies. The focus of the research has also shifted over time. Comparing expert and novice programmers was the focus early on in the research, although no common definition or measure of expertise was used in these studies. The focus has now shifted to comparing programmers with backgrounds in different programming paradigms and languages.

The contributions that resulted from the empirical research on program comprehension have been summarized in the Findings column of Table B.1. Although there are some researchers who build on their own work in an incremental fashion (Vessey, 1987) or use aspects of other studies as a model (Guerin and Matthews, 1990), this building process is limited and often deviates from the previous work in such a way that it is hard to connect the findings. Even studies that attempt to support the findings of other studies end up with inconclusive (Corritore and Wiedenbeck, 1999) or even contradictory results (Wiedenbeck and Ramalingam, 1999).

Another finding that emerged from the systematic review was that despite the widespread use of expert and novice as categories to describe programmers, no common definition or measure of expertise has been developed or adopted. Even studies that did not compare expert and novice programmers often used these categories to identify the group of programmers that were used as participants. There was also no consistency between the use of expert and experienced when describing programmers and these terms were often used interchangeably in the same study (Barfield, 1986; Vessey, 1987; Fix et al., 1993).

The present review contains empirical studies that have been used to build a timeline to provide insight as to how the research on program comprehension and mental representations has evolved. The collection of knowledge contained in the present review would be of interest to researchers who want to build on the work done by others in this field and those who want to expand this work to include new programming languages and paradigms. The present review demonstrates that the field of program comprehension is lacking incremental research that builds on previous work, and as a result, the research in this area has been scattered. The present review also points to gaps in the research on program comprehension. In recent years, the work in this field has declined dramatically and as a result, newly developed or popularized languages and paradigms have not been a part of the research reviewed here. In particular, parallel programming has been neglected in program comprehension research and consequently has developed using mostly informal approaches (Mattson and Wrinn, 2008). Because of the considerable differences between parallel programming and the programming examined in the studies in the present review, it is impossible to determine whether the findings summarized in Table B.1 would resemble the comprehension process and mental representations formed by parallel programmers. Therefore, future work should focus on empirical research designed to analyze the mental representations formed by expert parallel programmers during program comprehension in order to inform the development of tools and languages that support parallel programmers.

6. Future Work

The decline in research on mental representations is not an indication that this topic lacks relevance or importance, but an indication of a shift in focus. Research topics that have gained more interest include usability and comparative studies which allow tools and languages to be compared and ranked as more or less superior without the risk of finding that the tool or language does not coincide with the mental representation of the user. Ericsson et al. (2006) found that there has also been more focus on organizational settings where programmers work in teams (Gren et al., 2017; Teh et al., 2012; Dingsøyr and Dybå, 2012). The study of programming teams would be attractive to software companies as they may anticipate a more immediate return on their investment into this area of research. However, the use of mental model theory to design programming tools and languages has demonstrated measurable benefits. For example, Sulír (2015) studied mental model overlapping to develop source code annotations that allow programmers to share their mental models. From their study they found that the annotations improved program comprehension and reduced maintenance time. The program slicing tool developed by Korel and Rilling (1998) that assists with program comprehension was developed based on research that demonstrated the use of program slicing improves the process of program understanding. In our view, the current trend in which the usability of languages is assessed after they have already been in use is counterproductive as it ignores cognitive processes involved in programming. Accordingly, we argue that there needs to be a resurgence of empirical research on the psychology of programming to inform the design of tools and languages, especially in new and emerging programming paradigms.

The study of expert mental representations is important for informing the development of programming languages, instructional practices, and tools. To perform research on expert programmers it is necessary to be able to determine if participants are in fact experts. There has been a lack of agreement among researchers on how expertise should be measured. Siegmund et al. (2014) found that programmer experience can be determined by measuring their self estimation of their experience level compared to their peers and their experience level with logical programming. The study conducted by Baltes and Diehl (2018) found that self-assessment of expertise by programmers was not consistent between programmers with different programming language backgrounds and that years of experience was not related to expertise. The position taken by Parnin et al. (2017) is that expertise cannot be measured using superficial measures such as years of experience but instead using multiple measures such as observing the brain activity of programmers during program comprehension and assessing programming knowledge using concept inventories. To date there remains no standard for measuring programmer expertise so there is a need for more research to develop a standard measure for categorizing programmers based on their expertise. The distinction between expert and experienced also needs to be established. Experience is a measure of time spent working in a particular field or performing a task, however, it does not necessarily translate into expertise, which is a measure of performance (Ericsson et al., 2006). One recommendation for future work is to develop a tool for assessing programmer expertise that is not solely reliant on experience as a gauge.

Empirical research on mental representations formed by programmers during program comprehension has been predominately conducted using sequential code. Studies involving parallel programmers are most often concerned with productivity (Hochstein et al., 2005; Ebcioglu et al., 2006). However, the mental representations formed by expert parallel programmers during the comprehension of parallel programs is an important area of study in order to determine how their representations differ from the representations developed during sequential source code comprehension. The comprehension of parallel code requires programmers to mentally execute multiple timelines that are occurring in parallel at the machine level. Therefore, parallel program comprehension may require additional dimensions to construct a mental representation. To explore this research question, program comprehension studies need to be conducted using parallel programmers as participants and assign tasks that stimulate the comprehension process at a level that requires programmers to understand how the code executes in parallel. Possible tasks include identifying the presence of race conditions, rating efficiency or increasing efficiency of parallel programs.

The present literature review suggests that there is no consensus on the method that provides the most accurate account of the mental representations formed by programmers during the comprehension process. In addition, the different methods that have been used only provide an indirect analysis of these mental representations. The present review contains only one study, conducted by Fan (2010), that used eye tracking. Fan used eye tracking data to determine how the program comprehension process is affected by beacons, comments, and task motivation. This author concluded that eye tracking data can be used for tracing and analyzing the program comprehension process.

Work in psychology of programming has been moving towards the use of electroencephalography (EEG) to investigate models of cognition in recent years. For example, Crk et al. (2016) used EEG to determine programmer expertise. However, models of cognition are not the same as the mental representations that are of interest in the present review. In future work it is recommended that eye tracking be used in conjunction with direct questioning to formulate a model of the mental representations formed by programmers during program comprehension.

Acknowledgments

I would like to thank Richelle Witherspoon, Information Services Librarian at the University of New Brunswick for her assistance in formulating the search terms and guidance in executing the database searches.

Appendix A. Search String

The following search string was used for database keyword searches of Computer Source, ERIC, and PsycINFO. Modifications were made to this search string to fit the specific requirements for IEEE Xplore Digital Library and Scopus.

(((code OR program* OR software)N2(understanding OR comprehen* OR stud* OR analy* OR maint* OR modif* OR recall* OR sort* OR categor* OR debug* OR classif* OR *copy*)) AND (((cognit* OR mental* OR knowledge OR program* OR situation*)N2(model* OR represent* OR plan* OR structur* OR map* OR chunk* OR slic*)) OR schema*)) AND (programmer* OR coder*)

Appendix B. Summary of Literature

Table B.1: Summary of literature included in systematic review. Filled circles indicate that the feature applied and empty circles indicate that it did not apply. An extended table can be found at: `http://www.cs.unb.ca/~lbidlak1/ExtendedOnlineAppendix.pdf`

Author	Task											Method					Findings			
	Classify	Debug	Documentation	Enhancement	Hand Execution	Maintenance	Modification	Recopy	Reuse	Study	Write or Reconstruct Code	Comprehension Questions	Recall	Recognition	Verbal Protocol	Other	Chunking and Slicing	Mental Model Theory (multilevel)	Top Down vs. Bottom Up	Other
Shneiderman (1976)	○	○	○	○	●	○	○	○	○	●	○	●	●	○	○	○	●	○	○	○
Adelson (1981)	○	○	○	○	○	○	○	○	○	●	○	○	●	○	○	○	○	○	○	●
McKeithen et al. (1981)	○	○	○	○	○	○	○	○	○	●	○	○	●	○	○	○	●	○	○	○
Weiser (1981)	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	○	○	○
Weiser and Ledgard (1982)	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○	●	○	○	○
Adelson (1984)	○	●	○	○	○	○	○	○	○	●	○	●	○	○	○	○	○	○	○	●
Mynatt (1984)	○	○	○	○	●	○	○	○	○	●	○	○	●	○	○	○	●	○	○	○
Ehrlich and Soloway (1984)	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	●	●	○	○	○
Soloway and Ehrlich (1984)	○	○	○	○	○	○	○	○	○	●	○	○	●	○	○	●	○	○	○	●
Barfield (1986)	○	○	○	○	○	○	○	○	○	●	○	○	●	○	○	○	●	○	○	○
Schmidt (1986)	○	○	○	○	○	○	○	○	○	●	○	●	●	●	○	○	○	○	○	●
Bateson et al. (1987)	○	○	○	○	○	○	○	○	○	●	●	○	●	○	○	●	○	○	○	●
Boehm-Davis et al. (1987)	○	○	○	○	○	○	●	○	○	○	○	○	●	○	○	●	○	○	○	●
Letovsky (1987)	○	○	○	○	○	○	●	○	○	○	○	○	○	○	●	○	○	○	●	○
Littman et al. (1987)	○	○	○	●	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	●
Pennington (1987b)	○	○	○	○	○	●	○	○	○	●	○	●	○	●	○	○	○	●	○	○
Vessey (1987)	○	○	○	○	○	○	○	○	○	●	●	○	●	○	○	○	○	○	○	●
Détienne (1988)	○	○	○	○	○	○	○	○	○	●	○	○	○	○	●	○	○	○	●	○
Gilmore and Green (1988)	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	●
Vihmalo and Vihmalo (1988)	○	○	○	○	○	○	●	○	○	●	○	○	●	○	●	●	○	○	○	●
Davies (1990b)	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	●	○	○	○	●
Davies (1990a)	○	●	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	●

Continuation of Table B.1

Author	**Task**											**Method**					**Findings**			
	Classify	Debug	Documentation	Enhancement	Hand Execution	Maintenance	Modification	Recopy	Reuse	Study	Write or Reconstruct Code	Comprehension Questions	Recall	Recognition	Verbal Protocol	Other	Chunking and Slicing	Mental Model Theory (multilevel)	Top Down vs. Bottom Up	Other
Détienne and Soloway (1990)	○	○	○	○	○	○	○	○	○	●	○	○	○	○	●	●	○	○	○	●
Guerin and Matthews (1990)	○	○	○	○	○	○	○	○	○	●	○	○	●	○	○	●	●	○	○	○
Robertson and Yu (1990)	●	○	○	○	○	○	○	○	○	●	○	○	○	○	○	●	○	○	○	●
Bergantz and Hassell (1991)	○	○	○	○	○	○	●	○	○	●	○	○	○	○	●	○	○	●	○	○
Corritore and Wiedenbeck (1991)	○	○	○	○	○	○	○	○	○	●	○	●	○	○	○	●	○	○	●	○
Koenemann and Robertson (1991)	○	○	○	○	○	○	●	○	○	●	○	○	○	○	●	○	○	○	●	○
Koubek and Salvendy (1991)	○	○	○	○	○	○	●	○	○	○	○	○	○	○	●	○	○	○	○	●
Wiedenbeck (1991)	○	○	○	○	○	○	○	○	○	●	○	○	●	○	○	●	○	○	○	●
Boehm-Davis et al. (1992)	○	○	○	○	○	○	●	○	○	○	○	○	●	○	○	●	○	○	○	●
Fix et al. (1993)	○	○	○	○	○	○	○	○	○	●	○	●	○	○	○	○	○	○	○	●
von Mayrhauser and Vans (1993)	○	○	○	○	○	●	○	○	○	●	○	○	○	○	●	○	○	●	○	○
Wiedenbeck et al. (1993)	○	○	○	○	○	○	○	○	○	●	○	●	○	○	○	○	○	○	○	●
Davies (1994)	○	○	○	○	○	○	○	○	○	●	○	○	○	●	○	○	○	○	○	●
Teasley (1994)	○	○	○	○	○	○	○	○	○	●	○	●	○	○	○	○	○	○	●	○
von Mayrhauser and Vans (1994)	○	○	○	○	○	●	○	○	○	●	○	○	○	○	●	○	○	●	○	○
Burkhardt and Détienne	○	○	○	○	○	○	○	○	●	○	○	○	○	○	●	○	○	○	●	○
Davies et al. (1995)	●	○	○	○	○	○	○	○	○	●	○	○	○	○	○	●	○	○	○	●
Green and Navarro (1995)	○	○	○	○	○	○	○	○	○	●	○	●	○	○	○	●	○	○	○	●
Schömann (1995)	○	○	○	○	○	○	○	○	○	●	○	●	●	●	○	●	○	○	●	○
Shaft and Vessey (1995)	○	○	○	●	○	○	○	○	○	○	○	●	○	○	●	○	○	○	●	○
Snyder (1995)	○	○	○	○	○	○	●	○	○	○	○	●	○	○	●	○	○	○	○	●
von Mayrhauser and Vans (1995)	○	○	○	○	○	●	○	○	○	●	○	○	○	○	●	○	○	●	○	○
Vans (1996)	○	○	○	○	○	●	○	○	○	○	○	○	○	○	●	○	○	●	○	○
von Mayrhauser and Vans (1996)	○	○	○	○	○	●	○	○	○	●	○	○	○	○	●	○	○	●	○	○
Ye and Salvendy (1996)	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	●	○	○	●	○
Barfield (1997)	○	○	○	○	○	○	○	●	○	○	○	○	●	○	○	●	●	○	○	○
Burkhardt et al. (1997)	○	○	●	○	○	○	○	○	●	●	○	●	○	○	●	○	○	●	○	○
Ramalingam and Wiedenbeck (1997)	○	○	○	○	○	○	○	○	○	●	○	●	○	○	○	○	○	●	○	○

Continuation of Table B.1

Author	**Task**											**Method**					**Findings**			
	Classify	Debug	Documentation	Enhancement	Hand Execution	Maintenance	Modification	Recopy	Reuse	Study	Write or Reconstruct Code	Comprehension Questions	Recall	Recognition	Verbal Protocol	Other	Chunking and Slicing	Mental Model Theory (multilevel)	Top Down vs. Bottom Up	Other
von Mayrhauser et al. (1997)	○	○	○	●	○	○	○	○	○	○	○	○	○	○	●	○	○	●	○	○
Burkhardt et al. (1998)	○	○	●	○	○	○	○	○	●	●	○	○	○	○	●	●	○	○	●	○
Furman (1998)	○	○	○	○	○	○	○	○	○	●	○	●	○	○	○	●	●	○	○	○
Shaft and Vessey (1998)	○	○	○	●	○	○	○	○	○	○	○	●	○	○	●	○	○	○	●	○
von Mayrhauser and Vans (1998)	○	○	○	○	○	●	○	○	○	○	○	○	○	○	●	○	○	●	○	○
Wong et al. (1998)	○	○	○	○	○	○	○	○	○	●	○	○	○	●	○	○	○	○	○	●
Corritore and Wiedenbeck (1999)	○	○	○	○	○	●	○	○	○	●	○	●	○	○	○	○	○	●	○	○
Vans et al. (1999)	○	○	○	○	○	●	○	○	○	○	○	○	○	○	●	○	○	●	○	○
Wiedenbeck and Ramalingam (1999)	○	○	○	○	○	○	○	○	○	●	○	●	○	○	○	○	○	●	○	○
Wiedenbeck et al. (1999)	○	○	○	○	○	○	○	○	○	●	○	●	○	○	○	○	○	●	○	○
Corritore and Wiedenbeck (2001)	○	○	○	○	○	●	○	○	○	●	○	○	○	○	○	●	○	○	●	○
Mosemann and Wiedenbeck (2001)	○	○	○	○	○	○	○	○	○	●	○	●	○	○	○	○	○	○	●	○
Navarro-Prieto and Cañas (2001)	○	○	○	○	○	○	●	○	○	●	○	○	○	●	○	●	○	○	○	●
Romero (2001)	○	○	○	○	○	○	○	○	○	●	○	●	●	●	○	○	○	○	○	●
Burkhardt et al. (2002)	○	○	●	○	○	○	○	○	●	●	○	●	○	○	○	○	○	●	○	○
Khazaei and Jackson (2002)	○	○	○	○	○	○	○	○	○	●	○	●	○	○	○	○	○	○	○	●
Parkin (2004)	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	●	○	●	○	○
Romero and Du Boulay (2004)	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	●
Sajaniemi and Prieto (2005)	●	○	○	○	○	○	●	○	○	●	○	○	○	○	○	●	○	○	○	●
Fan (2010)	○	●	○	○	○	○	○	○	○	●	○	○	●	○	○	●	●	○	○	○
Alardawi and Agil (2015)	○	○	○	○	○	○	○	○	○	●	○	●	○	○	○	○	○	○	○	●
Nosál and Porubän (2015)	○	○	○	○	○	○	●	○	○	●	○	●	○	○	●	○	○	○	○	●

End of Table

References

Adelson, B., 1981. Problem solving and the development of abstract categories in programming languages. Memory & Cognition 9, 422–433. doi:10.3758/BF03197568.

Adelson, B., 1984. When novices surpass experts: The difficulty of a task may increase with expertise. Journal of Experimental Psychology: Learning, Memory, and Cognition 10, 483–495. doi:10.1037/0278-7393.10.3.483.

Alardawi, A., Agil, A., 2015. Novice comprehension of object-oriented oo programs: An empirical study, in: 2015 World Congress on Information Technology and Computer Applications (WCITCA), pp. 1–4. doi:10.1109/WCITCA.2015.7367057.

Arab, M., 1992. Enhancing program comprehension: Formatting and documenting. ACM SIGPLAN Notices 27, 37–46. doi:10.1145/130973.130975.

Atwood, M.E., Ramsey, H.R., 1978. Cognitive structures in the comprehension and memory of computer programs: An investigation of computer program debugging. Technical Report. SCIENCE APPLICATIONS INC ENGLEWOOD CO.

Baltes, S., Diehl, S., 2018. Towards a theory of software development expertise, in: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ACM, New York, NY, USA. pp. 187–200. doi:10.1145/3236024.3236061.

Barfield, W., 1986. Expertnovice differences for software: Implications for problem-solving and knowledge acquisition. Behaviour & Information Technology 5, 15–29. doi:10.1080/01449298608914495.

Barfield, W., 1997. Skilled performance on software as a function of domain expertise and program organization. Perceptual and Motor Skills 85, 1471–1480. doi:10.2466/pms.1997.85.3f.1471.

Basili, V.R., Reiter, R.W., 1981. A controlled experiment quantitatively comparing software development approaches. IEEE Transactions on Software Engineering 7, 299–320. doi:10.1109/TSE.1981.230841.

Bateson, A.G., Alexander, R.A., Murphy, M.D., 1987. Cognitive processing differences between novice and expert computer programmers. International Journal of Man-Machine Studies 26, 649–660. doi:10.1016/S0020-7373(87)80058-5.

Bergantz, D., Hassell, J., 1991. Information relationships in prolog programs: How do programmers comprehend functionality?. International Journal of Man-Machine Studies 35, 313–328. doi:10.1016/S0020-7373(05)80131-2.

Boehm-Davis, D.A., Holt, R.W., Schultz, A.C., 1987. Development and use of cognitive representations of software in a modification task., in: Proceedings of the 1987 IEEE International Conference on Systems, Man and Cybernetics, pp. 990–994.

Boehm-Davis, D.A., Holt, R.W., Schultz, A.C., 1992. The role of program structure in software maintenance. International Journal of Man-Machine Studies 36, 21–63. doi:10.1016/0020-7373(92)90051-L.

Boshernitsan, M., Graham, S.L., Hearst, M.A., 2007. Aligning development tools with the way programmers think about code changes, in: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM. pp. 567–576. doi:10.1145/1240624.1240715.

Budgen, D., Brereton, P., 2006. Performing systematic literature reviews in software engineering, in: Proceedings of the 28th International Conference on Software Engineering, ACM. pp. 1051–1052. doi:10.1145/1134285.1134500.

Burkhardt, J., Détienne, F., Wiedenbeck, S., 1998. The effect of object-oriented programming expertise in several dimensions of comprehension strategies, in: Proceedings. 6th International Workshop on Program Comprehension. IWPC98 (Cat. No.98TB100242), pp. 82–89. doi:10.1109/WPC.1998.693294.

Burkhardt, J.M., Détienne, F., . An empirical study of software reuse by experts in object-oriented design, in: HumanComputer Interaction: Interact 95. Springer US. IFIP Advances in Information and Communication Technology, pp. 133–138. doi:10.1007/978-1-5041-2896-4_22.

Burkhardt, J.M., Détienne, F., Wiedenbeck, S., 1997. Mental representations constructed by experts and novices in object-oriented program comprehension, in: Human-Computer Interaction INTERACT 97, Springer US. pp. 339–346. doi:10.1007/978-0-387-35175-9_55.

Burkhardt, J.M., Détienne, F., Wiedenbeck, S., 2002. Object-oriented program comprehension: Effect of expertise, task and phase. Empirical Software Engineering 7, 115–156. doi:10.1023/A:1015297914742.

Cañas, J.J., Antolí, A., 1998. The role of working memory in measuring mental models, in: Proceedings of the Ninth European Conference on Cognitive Ergonomics–Cognition and Cooperation. European Association of Cognitive Ergonomics (EACE): Rocquencourt, France.

Chamberlain, B.L., Deitz, S.J., Snyder, L., 2000. A comparative study of the NAS MG benchmark across parallel languages and architectures, in: Supercomputing, ACM/IEEE 2000 Conference, pp. 46–46. doi:10.1109/SC.2000.10006.

Corritore, C., Wiedenbeck, S., 1991. What do novices learn during program comprehension? International Journal of Human-Computer Interaction 3, 199–222. doi:10.1080/10447319109526004.

Corritore, C.L., Wiedenbeck, S., 1999. Mental representations of expert procedural and object-oriented programmers in a software maintenance task. International Journal of Human-Computer Studies 50, 61–83. doi:10.1006/ijhc.1998.0236.

Corritore, C.L., Wiedenbeck, S., 2001. An exploratory study of program comprehension strategies of procedural and object-oriented programmers. International Journal of Human-Computer Studies 54, 1–23. doi:10.1006/ijhc.2000.0423.

Crk, I., Kluthe, T., Stefik, A., 2016. Understanding programming expertise: An empirical study of phasic brain wave changes. ACM Transactions on Computer-Human Interaction 23, 1–29. doi:10.1145/2829945.

Davies, S., 1990a. The nature and development of programming plans. International Journal of Man-Machine Studies 32, 461–481. doi:10.1016/S0020-7373(05)80143-9.

Davies, S.P., 1990b. Plans, goals and selection rules in comprehension of computer programs. Behaviour & Information Technology 9, 201–214. doi:10.1080/01449299008924237.

Davies, S.P., 1994. Knowledge restructuring and the acquisition of programming expertise. International Journal of Human-Computer Studies 40, 703–726. doi:10.1006/ijhc.1994.1032.

Davies, S.P., Gilmore, D.J., Green, T.R.G., 1995. Are objects that important? Effects of expertise and familiarity on classification of object-oriented code. Human-Computer Interaction 10, 227–248. doi:10.1207/s15327051hci1002\&3_3.

Détienne, F., 1988. Une application de la théorie des schémas à la compréhension de programmes. = Applying schema theory to program understanding. Le Travail Humain: A Bilingual and Multi-Disciplinary Journal in Human Factors 51, 335–350.

Détienne, F., 2001. Software Design-Cognitive Aspect. Springer Science & Business Media.

Détienne, F., Soloway, E., 1990. An empirically-derived control structure for the process of program understanding. International Journal of Man-Machine Studies 33, 323–342. doi:10.1016/S0020-7373(05)80122-1.

Dillon, E.C.J., 2013. Measuring the effects of low assistive vs. moderately assistive environments on novice programmers. Ph.D. thesis. ProQuest Information & Learning.

Dingsøyr, T., Dybå, T., 2012. Team effectiveness in software development: Human and cooperative aspects in team effectiveness models and priorities for future studies, in: 2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering (CHASE), pp. 27–29. doi:10.1109/CHASE.2012.6223016.

Ebcioglu, K., Sarkar, V., El-Ghazawi, T., Urbanic, J., Center, P.S., 2006. An experiment in measuring the productivity of three parallel programming languages, in: Proceedings of the Third Workshop on Productivity and Performance in High-End Computing, pp. 30–36.

Ehrlich, K., Soloway, E., 1984. An empirical investigation of the tacit plan knowledge in programming, in: Human factors in computer systems, pp. 113–134.

Ericsson, K., Charness, N.E., Feltovich, P.J., Hoffman, R.R., 2006. The Cambridge Handbook of Expertise and Expert Performance. Cambridge Handbooks in Psychology, Cambridge University Press. doi:10.1017/CBO9780511816796.

Fan, Q., 2010. The effects of beacons, comments, and tasks on program comprehension process in software maintenance. Ph.D. thesis.

Feo, J.T., 2015. A Comparative Study of Parallel Programming Languages: the Salishan Problems. Special Topics in Supercomputing, v. 6, Elsevier Science.

Fix, V., Wiedenbeck, S., Scholtz, J., 1993. Mental representations of programs by novices and experts, in: Conference on Human Factors in Computing Systems - Proceedings, pp. 74–79.

Furman, S.M., 1998. Improving software comprehension. Ph.D. thesis. ProQuest Information & Learning.

Gilmore, D.J., Green, T.R., 1988. Programming plans and programming expertise. The Quarterly Journal of Experimental Psychology A: Human Experimental Psychology 40, 423–442. doi:10.1080/02724988843000005.

Green, T.R.G., 1989. Cognitive dimensions of notations, in: Proceedings of the Fifth Conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and Computers V, Cambridge University Press, New York, NY, USA. pp. 443–460.

Green, T.R.G., Navarro, R., 1995. Programming plans, imagery, and visual programming, in: HumanComputer Interaction: Interact 95. Springer US. IFIP Advances in Information and Communication Technology, pp. 139–144. doi:10.1007/978-1-5041-2896-4_23.

Gren, L., Torkar, R., Feldt, R., 2017. Group development and group maturity when building agile teams: A qualitative and quantitative investigation at eight large companies. Journal of Systems and Software 124, 104–119. doi:10.1016/j.jss.2016.11.024.

Guerin, B., Matthews, A., 1990. The effects of semantic complexity on expert and novice computer program recall and comprehension. Journal of General Psychology 117, 379–389. doi:10.1080/00221309.1990.9921144.

Hochstein, L., Basili, V.R., Vishkin, U., Gilbert, J., 2008. A pilot study to compare programming effort for two parallel programming models. Journal of Systems & Software 81, 1920–1930. doi:10.1016/j.jss.2007.12.798.

Hochstein, L., Carver, J., Shull, F., Asgari, S., Basili, V., Hollingsworth, J.K., Zelkowitz, M.V., 2005. Parallel programmer productivity: A case study of novice parallel programmers, in: Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference, pp. 35–35. doi:10.1109/SC.2005.53.

Hornbæk, K., 2006. Current practice in measuring usability: Challenges to usability studies and research. International Journal of Human-Computer Studies 64, 79–102. doi:https://doi.org/10.1016/j.ijhcs.2005.06.002.

Kamma, D., Jalote, P., 2013. Effect of task processes on programmer productivity in model-based testing, in: Proceedings of the 6th India software engineering conference, ACM. pp. 23–28. doi:10.1145/2442754.2442758.

Khazaei, B., Jackson, M., 2002. Is there any difference in novice comprehension of a small program written in the event-driven and object-oriented styles?, in: Proceedings - IEEE 2002 Symposia on Human Centric Computing Languages and Environments, HCC 2002, pp. 19–26.

Koenemann, J., Robertson, S.P., 1991. Expert problem solving strategies for program comprehension, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM. pp. 125–130. doi:10.1145/108844.108863.

Korel, B., Rilling, J., 1998. Program slicing in understanding of large programs, in: Proceedings. 6th International Workshop on Program Comprehension. IWPC98 (Cat. No.98TB100242), pp. 145–152. doi:10.1109/WPC.1998.693339.

Koubek, R.J., Salvendy, G., 1991. Cognitive performance of super-experts on computer program modification tasks. Ergonomics 34, 1095–1112. doi:10.1080/00140139108964849.

Landman, D., Serebrenik, A., Vinju, J.J., 2017. Challenges for static analysis of java reflection-literature review and empirical study, in: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), IEEE. pp. 507–518.

Lane, H.C., 2005. Natural language tutoring and the novice programmer. Ph.D. thesis. ProQuest Information & Learning.

Letovsky, S., 1987. Cognitive processes in program comprehension. The Journal of Systems and Software 7, 325–339. doi:10.1016/0164-1212(87)90032-X.

Littman, D., Pinto, J., Letovsky, S., Soloway, E., 1987. Mental models and software maintenance. The Journal of Systems and Software 7, 341–355. doi:10.1016/0164-1212(87)90033-1.

Mattson, T., Wrinn, M., 2008. Parallel programming: Can we please get it right this time?, in: Proceedings of the 45th Annual Design Automation Conference, ACM, New York, NY, USA. pp. 7–11. doi:10.1145/1391469.1391474.

von Mayrhauser, A., Vans, A., 1994. Comprehension processes during large scale maintenance, in: Proceedings of 16th International Conference on Software Engineering, pp. 39–48. doi:10.1109/ICSE.1994.296764.

von Mayrhauser, A., Vans, A., 1995. Industrial experience with an integrated code comprehension model. Software engineering journal 10, 171–182. doi:10.1049/sej.1995.0023.

von Mayrhauser, A., Vans, A., 1996. Identification of dynamic comprehension processes during large scale maintenance. IEEE Transactions on Software Engineering 22, 424–437. doi:10.1109/32.508315.

von Mayrhauser, A., Vans, A.M., 1993. From code understanding needs to reverse engineering tool capabilities, in: Proceedings of 6th International Workshop on Computer-Aided Software Engineering, pp. 230–239. doi:10.1109/CASE.1993.634824.

von Mayrhauser, A., Vans, A.M., 1998. Program understanding behavior during adaptation of large scale software, in: Proceedings. 6th International Workshop on Program Comprehension. IWPC98 (Cat. No.98TB100242), pp. 164–172. doi:10.1109/WPC.1998.693345.

von Mayrhauser, A., Vans, A.M., Howe, A.E., 1997. Program understanding behaviour during enhancement of large-scale software. Journal of Software Maintenance: Research and Practice 9, 299–327. doi:10.1002/(SICI)1096-908X(199709/10)9:5<299::AID-SMR157>3.0.CO;2-S.

McKeithen, K., Reitman, J., Rueter, H., Hirtle, S., 1981. Knowledge organization and skill differences in computer programmers. Cognitive Psychology 13, 307–325. doi:10.1016/0010-0285(81)90012-8.

Moher, D., Liberati, A., Tetzlaff, J., Altman, D.G., Group, T.P., 2009. Preferred reporting items for systematic reviews and meta-analyses: The prisma statement. PLOS Medicine 6. doi:10.1371/journal.pmed.1000097.

Mosemann, R., Wiedenbeck, S., 2001. Navigation and comprehension of programs by novice programmers, in: Proceedings 9th International Workshop on Program Comprehension. IWPC 2001, IEEE. pp. 79–88.

Mynatt, B.T., 1984. The effect of semantic complexity on the comprehension of program modules. International Journal of Man-Machine Studies 21, 91–103. doi:10.1016/S0020-7373(84)80060-7.

Nanz, S., Furia, C.A., 2015. A comparative study of programming languages in rosetta code, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, pp. 778–788. doi:10.1109/ICSE.2015.90.

Navarro-Prieto, R., Cañas, J.J., 2001. Are visual programming languages better? The role of imagery in program comprehension. International Journal of Human-Computer Studies 54, 799–829. doi:10.1006/ijhc.2000.0465.

Nosál, M., Porubän, J., 2015. Program comprehension with four-layered mental model, in: 2015 13th International Conference on Engineering of Modern Electric Systems (EMES), pp. 1–4. doi:10.1109/EMES.2015.7158420.

Oliveira Aureliano, V.C., 2013. A methodology for teaching programming for beginners, in: Proceedings of the ninth annual international ACM conference on International computing education research, ACM. pp. 169–170.

Pariser, E., 2011. The Filter Bubble: How the New Personalized Web Is Changing What We Read and How We Think. Penguin. Google-Books-ID: wcalrOI1YbQC.

Parkin, P., 2004. An exploratory study of code and document interactions during task-directed program comprehension, in: 2004 Australian Software Engineering Conference. Proceedings., pp. 221–230. doi:10.1109/ASWEC.2004.1290475.

Parnin, C., Siegmund, J., Peitek, N., 2017. On the nature of programmer expertise., in: PPIG, p. 16.

Pennington, N., 1987a. Comprehension strategies in programming, in: Empirical Studies of Programmers: Second Workshop, Ablex Publishing Corp.. pp. 100–113.

Pennington, N., 1987b. Stimulus structures and mental representations in expert comprehension of computer programs. Cognitive Psychology 19, 295–341. doi:10.1016/0010-0285(87)90007-7.

Petre, M., Blackwell, A.F., 1999. Mental imagery in program design and visual programming. International Journal of Human-Computer Studies 51, 7–30. doi:10.1006/ijhc.1999.0267.

Prechelt, L., 2000. An empirical comparison of seven programming languages. Computer 33, 23–29. doi:10.1109/2.876288.

Ramalingam, V., Wiedenbeck, S., 1997. An empirical study of novice program comprehension in the imperative and object-oriented styles, in: Papers Presented at the 7th Workshop on Empirical Studies of Programmers, ESP 1997, pp. 124–139.

Robertson, S., Yu, C.C., 1990. Common cognitive representations of program code across tasks and languages. International Journal of Man-Machine Studies 33, 343–360. doi:10.1016/S0020-7373(05)80123-3.

Romero, P., 2001. Focal structures and information types in prolog. International Journal of Human-Computer Studies 54, 211236. doi:10.1006/

ijhc.2000.0408.
Romero, P., Du Boulay, B., 2004. Structural knowledge and language notational properties in program comprehension, in: 2004 IEEE Symposium on Visual Languages - Human Centric Computing, pp. 223–225. doi:10.1109/VLHCC.2004.50.
Sajaniemi, J., Prieto, R.N., 2005. An investigation into professional programmers' mental representations of variables, in: Proceedings - IEEE Workshop on Program Comprehension, pp. 55–64.
Schmidt, A.L., 1986. Effects of experience and comprehension on reading time and memory for computer programs. International Journal of Man-Machine Studies 25, 399–409. doi:10.1016/S0020-7373(86)80068-2.
Scholtz, J., Wiedenbeck, S., 1990. Learning second and subsequent programming languages: A problem of transfer. International Journal of Human-Computer Interaction 2, 51–72. doi:10.1080/10447319009525970.
Schömann, M., 1995. Knowledge organization of novices and advanced programmers: Effect of previous knowledge on recall and recognition of lisp-procedures, in: Cognition and computer programming, Ablex Publishing. pp. 193–217.
Shaft, T.M., Vessey, I., 1995. Research report-the relevance of application domain knowledge: The case of computer program comprehension. Info. Sys. Research 6, 286–299. doi:10.1287/isre.6.3.286.
Shaft, T.M., Vessey, I., 1998. The relevance of application domain knowledge: Characterizing the computer program comprehension process. Journal of Management Information Systems 15, 51–78. doi:10.1080/07421222.1998.11518196.
Shneiderman, B., 1976. Exploratory experiments in programmer behavior. International Journal of Computer & Information Sciences 5, 123–143. doi:10.1007/BF00975629.
Siegmund, J., Kästner, C., Liebig, J., Apel, S., Hanenberg, S., 2014. Measuring and modeling programming experience. Empirical Software Engineering 19, 1299–1334. doi:10.1007/s10664-013-9286-4.
Snyder, J.R., 1995. The role of local program information in software maintenance productivity. Ph.D. thesis. ProQuest Information & Learning.
Soloway, E., Ehrlich, K., 1984. Empirical studies of programming knowledge. IEEE Transactions on Software Engineering SE-10, 595–609. doi:10.1109/TSE.1984.5010283.
Stone, D.N., Jordan, E.W., Wright, M.K., 1990. The impact of pascal education on debugging skill. International Journal of Man-Machine Studies 33, 81–95. doi:10.1016/S0020-7373(05)80116-6.
Storey, M.A., 2006. Theories, tools and research methods in program comprehension: past, present and future. Software Quality Journal 14, 187–208. doi:10.1007/s11219-006-9216-4.
Sulír, M., 2015. Sharing developers mental models through source code annotations, in: 2015 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 997–1006. doi:10.15439/2015F301.
Teasley, B., 1994. The effects of naming style and expertise on program comprehension. International Journal of Human - Computer Studies 40, 757–770. doi:10.1006/ijhc.1994.1036.
Teh, A., Baniassad, E., Rooy, D.v., Boughton, C., 2012. Social psychology and software teams: Establishing task-effective group norms. IEEE Software 29, 53–58. doi:10.1109/MS.2011.157.
Tubaishat, A., 2001. A knowledge base for program debugging, in: AICCSA '01 Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications, pp. 321–327. doi:10.1109/AICCSA.2001.934005.
Vans, A.M., 1996. A multi-level code comprehension model for large-scale software. Ph.D. thesis. ProQuest Information & Learning.
Vans, A.M., von Maryhauser, A., Somlo, G., 1999. Program understanding behavior during corrective maintenance of large-scale software. International Journal of Human-Computer Studies 51, 31–70. doi:10.1006/ijhc.1999.0268.
Vessey, I., 1987. On matching programmers' chunks with program structures: An empirical investigation. International Journal of Man-Machine Studies 27, 65–89. doi:10.1016/S0020-7373(87)80044-5.
Vihmalo, A., Vihmalo, M., 1988. Utilization of subject's background knowledge in computer program comprehension. Zeitschrift fr Psychologie mit Zeitschrift fr angewandte Psychologie 196, 401–413.
Weiser, M., 1981. Program slicing, in: Proceedings - International Conference on Software Engineering, pp. 439–449.
Weiser, M., Ledgard, H., 1982. Programmers use slices when debugging. Communications of the ACM 25, 446–452. doi:10.1145/358557.358577.
Whalley, J., Kasto, N., 2014. A qualitative think-aloud study of novice programmers code writing strategies, in: Proceedings of the 2014 conference on Innovation & Technology in Computer Science Education, pp. 279–284. doi:10.1145/2591708.2591762.
Wiedenbeck, S., 1991. The initial stage of program comprehension. International Journal of Man-Machine Studies 35, 517–540. doi:10.1016/S0020-7373(05)80090-2.
Wiedenbeck, S., Fix, V., Scholtz, J., 1993. Characteristics of the mental representations of novice and expert programmers: An empirical study. International Journal of Man-Machine Studies 39, 793–812. doi:10.1006/imms.1993.1084.
Wiedenbeck, S., Ramalingam, V., 1999. Novice comprehension of small programs written in the procedural and object-oriented styles. International Journal of Human-Computer Studies 51, 71–87. doi:10.1006/ijhc.1999.0269.
Wiedenbeck, S., Ramalingam, V., Sarasamma, S., Corritore, C.L., 1999. A comparison of the comprehension of object-oriented and procedural programs by novice programmers. Interacting with Computers 11, 255–282. doi:10.1016/S0953-5438(98)00029-0.
Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th international conference on evaluation and assessment in software engineering, Citeseer. p. 38.
Wong, S.Y., Cheung, H., Chen, H.C., 1998. The advanced programmers reliance on program semantics: Evidence from some cognitive tasks. International Journal of Psychology 33, 259268. doi:10.1080/002075998400303.
Ye, N., Salvendy, G., 1996. An objective approach to exploring skill differences in strategies of computer program comprehension. Behaviour & Information Technology 15, 139–148. doi:10.1080/014492996120229.
Yeh, K.C., 2014. Toward understanding the cognitive processes of software design in novice programmers. Ph.D. thesis. ProQuest Information & Learning.