Pilot Study: Validation of Stimuli for Studying Mental Representations Formed by Parallel Programmers During Parallel Program Comprehension

Leah Bidlake **Eric Aubanel** **Daniel Voyer**
Faculty of Computer Science, Faculty of Computer Science, Department of Psychology
University of New Brunswick
leah.bidlake@unb.ca, aubanel@unb.ca, voyer@unb.ca

Abstract

Research on mental representations formed by programmers during program comprehension has not yet been applied to parallel programming. The goals of the pilot study were to validate a stimulus set, consisting of 80 programs written in C using OpenMP 4.0 directives, that will be used in subsequent studies on mental representations formed by expert parallel programmers and to serve as a resource for researchers who want to replicate or expand the research on program comprehension to include the parallel programming paradigm. The task used to stimulate the comprehension process was determining the presence of data races. Responses to the data race question were analyzed to determine the validity of the stimuli.

The results of the pilot study indicate that the level of difficulty of the stimuli (accuracy rate of .65) and the time limit for exposure to the stimuli are both appropriate and do not need to be adjusted for the main study. Participants' self-perceived level of expertise correlated with their accuracy indicating this is a reasonable measure of expertise. Given the disparity of responses when asking participants what cues or program components they used to determine whether or not there was a data race, the main study will also include specific questions about components of the code to determine the type of information that is included in their mental representations.

1. Introduction

During the comprehension process, programmers form mental representations of the code they are working with (Détienne, 2001). Understanding these representations is important for developing programming languages and tools that enhance and assist programmers in the comprehension process and other tasks. 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). 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).

Parallel programming has introduced new challenges including bugs that are hard to detect, making it difficult for programmers to verify correctness of code. For example, data races are a type of bug that can occur only in parallel programming and their detection often require close consideration of the code. Data races occur when multiple threads of execution access the same memory location without controlling the order of the accesses and at least one of the memory accesses is a write (Liao, Lin, Asplund, Schordan, & Karlin, 2017). Depending on the order of the accesses, some threads may read the memory location before the write and others may read the memory location after the write, which can lead to unpredictable results and incorrect program execution. Data races are difficult to detect and verify as they will not appear every time that the program is executed. To detect data races, programmers must understand how a program executes in parallel on the machine and the memory model of the programming language.

In parallel programming, there is a significant lack of theory to inform the development of programming languages, instructional practices, and tools (Mattson & Wrinn, 2008). Empirical research on

mental representations formed by programmers during program comprehension has been predominately conducted using sequential code. 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.

2. Background

Empirical research on program comprehension has a direct impact on the development of programming languages and tools. For example, tools have been developed to assist programmers with maintaining, debugging, and documenting code using principles of program comprehension. For instance, Boshernitsan, Graham, and Hearst (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 T. R. G. Green (1989) to provide programmers with visual representations that reflect their own mental model of the source code. Empirical research on programming knowledge and plans was used by Tubaishat (2001) to develop a theoretical model, Conceptual Model for Software Fault Localization (CMSFL). 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. Another example is a tool developed by Arab (1992) 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.

In terms of task parameters, program comprehension studies have used a wide variety of code constructs ranging from code snippets with as few as eight lines of code (Gilmore & Green, 1988) to complete industrial code (von Mayrhauser & Vans, 1998). There are a number of studies that have used programs with fewer than 20 lines of code (Davies, 1990; Furman, 1998; Gilmore & Green, 1988; Soloway & Ehrlich, 1984), and between 20-30 lines of code (Barfield, 1997; Bateson, Alexander, & Murphy, 1987; Bergantz & Hassell, 1991; Pennington, 1987b; Ramalingam, LaBelle, & Wiedenbeck, 2004; Shargabi, Aljunid, Annamalai, & Zin, 2020; Teasley, 1994; Wiedenbeck & Ramalingam, 1999). Time limits ranging from 60 to 120 seconds have been used in program comprehension experiments with programs that range between eight and 28 lines of code (Gilmore & Green, 1988; Pennington, 1987b; Ramalingam et al., 2004; Teasley, 1994; Wiedenbeck & Ramalingam, 1999). Research in program comprehension has also been conducted using larger programs and software systems that more accurately reflect real life situations involving thousands of lines of code (Abbes, Khomh, Guéhéneuc, & Antoniol, 2011; Bavota et al., 2013; Nosál' & Porubän, 2015). These studies tend to use very few stimuli and often involve a small number of participants providing little power.

3. Research Goals

To date, no empirical research on program comprehension or mental representations of parallel programmers has been conducted (Bidlake, Aubanel, & Voyer, 2020). Because of the considerable differences between parallel and sequential programming, it is impossible to determine if the findings of the empirical research using sequential code would resemble the comprehension process and mental representations of parallel programmers. To inform the development of tools and languages that specifically support parallel programmers, it is important to analyze the mental representations formed by parallel programmers during program comprehension. Therefore, empirical research on program comprehension needs to be expanded to include parallel programming. The lack of research in this programming paradigm also means that there are no existing data sets or stimuli to draw from. The research goal of the pilot study was to validate the stimulus set we created as producing a reasonable level of difficulty.

The long term research goal is to develop a model for parallel program comprehension that is based on the abstract mental representations formed by parallel programmers during program comprehension. Our future studies to investigate these models will make use of the stimuli developed here. The first step in realizing this goal will be to conduct a study to investigate the progression of mental models formed by programmers during instruction on parallel programming.

The stimuli we developed would be relevant for replication studies and incremental research that builds on previous work in the psychology of programming field. The stimuli would also be useful for those who want to expand the research on program comprehension to include the parallel programming paradigm.

4. Method

We conducted an online pilot study with eight participants. The results of the pilot study were analyzed to determine if any of the parameters need to be adjusted for the main study including exposure duration and difficulty of stimuli.

4.1. Participants

Participants had to have experience programming in C and using OpenMP 4.0 directives to implement parallelization. To recruit participants, university instructors emailed the advertisement to students who had completed their parallel programming course and colleagues that would have the appropriate background to complete the study. In the end, a final sample of eight participants completed the experiment. Five participants were professionals and three participants were students. The mean age of participants was 30 years. Their mean amount of programming experience was 8 years. Participants could choose to receive a $10 e-gift card as an incentive. Participants were informed in the consent form that the incentive is only available in select countries. Participation was voluntary and the protocol was approved by the research ethics board at UNB.

4.2. Materials

The programs from the DataRaceBench 1.2.0 benchmark suite (Liao et al., 2017) were used as inspiration for the programs written by the first two authors, who are computer science instructors. The stimuli were all programs written in C using OpenMP 4.0 directives with no comments or documentation. The selection of OpenMP directives for the stimuli was based largely on the set of directives referred to as The Common Core (Mattson, Koniges, He, & Chapman, 2018).

Research in program comprehension has shown that programmers possess programming plan knowledge consisting of typical solutions to problems, such as searching for a value in a data structure (Soloway & Ehrlich, 1984). The programs for the stimuli were written using unplan-like code so that participants would have to construct their mental representation without relying on prior plan knowledge. We selected the task of detecting data races as it requires programmers to mentally execute the code at the machine level and consider how the execution occurs in multiple timelines in parallel. This additional layer of complexity that requires considering multiple timelines of execution and how they interact will likely result in mental representations that differ from the program and situation models formed during program comprehension of sequential code.

There were multiple considerations when determining the number of stimuli and the length of each stimulus. Given that we are sampling a very specific population of programmers that must have experience using OpenMP directives in C, we did not expect to be able to recruit a large number of participants for the main study. To ensure adequate power with a potentially small number of participants, we had to have a large number of stimuli to meet the minimum recommendation proposed by Brysbaert and Stevens (2018). We also wanted to make sure that the time to complete the experiment was approximately one hour so that it would be a reasonable request for busy professionals. Another consideration was that our stimuli would be used in future experiments using an eye tracking device. To reduce the complexity of eye tracking, we wanted to ensure that the stimuli would fit on a single screen. Given the extensive research that has been done in program comprehension using small scale programs, between eight and 30 lines of code (see section 2), we developed stimuli that ranged between 17 and 24 lines of code to meet the aforementioned needs of our study. We also wanted to use a variety of OpenMP directives including those from the common core to account for differences in participant background knowledge (i.e.: participants may be more familiar with some directives and not others). Using 13 different directives and writing multiple programs using each directive with an equal number of programs with and without a data race, we were able to create 80 unique programs all containing a parallel region.

We felt this was an adequate number of stimuli to draw from for our pilot and main study so that if it was found that some of the stimuli were too difficult or if participants performed poorly on particular directives, we would still have a substantial data set for future research.

The purpose of a time limit for exposure was to determine a reasonable amount of time for participants to complete the task without giving them additional time to read the code for other purposes. To study the mental representations of participants, specific questions about components of the code can be used to determine the type of information they have in working memory and would likely be part of their mental model. Pennington (1986) found that the task (e.g.: study, modification) influenced programmers' mental models. Our concern is that once participants are asked questions pertaining to specific parts of the code that may not be part of their model they may then use additional time to study the code in order to prepare for these questions, creating a practice effect. The time limits used in program comprehension studies, with both novice and expert participants, have ranged between 60 and 120 seconds using programs of similar length to ours (see section 2). Given that our target population is experts, we selected a time limit of 60 seconds.

To reduce the mental strain of tracing code, variable names used in the stimuli match typical programming conventions (e.g.: variables i, j, and k are used for loop counters) (Beniamini, Gingichashvili, Orbach, & Feitelson, 2017) and the variable names were consistent between stimuli to reduce the mental load (e.g.: variables used for arrays were a, b, and c, the variables used for the size of the arrays were n and m).

Four of the stimuli were used as practice and one practice stimulus was followed by the question asking the participant what cues or components they used to determine whether or not there was a data race. The practice allowed participants to familiarize themselves with all aspects of the interface (e.g.: use of the visual analogue scale and entering text) and the ratio of stimuli followed by the question in the practice (25%) is similar to that of the experiment (26%). For the 76 stimuli used for the experiment, 38 of the stimuli contained a data race and 38 of the stimuli did not (see Figure 1). The length of the stimuli was measured by the number of lines of code. Although we recognize that the lines of code metric does not necessarily reflect the complexity of the code, especially in parallel programming, it is a metric that is commonly used for just that (Bhatia & Malhotra, 2014). Therefore, it is reasonable to expect that programmers would also initially judge the complexity of the programs based on their length. In order to mitigate this we made our programs similar in length. When counting the lines of code we only excluded blank lines, and therefore included all lines that contained any text or symbols. The length of the stimuli with a data race (mean = 20.95, SD = 1.52) and the length of the stimuli without a data race (mean = 20.95, SD = 1.29) did not differ significantly from each other ($p = 1$).

4.3. Procedure

Participants completed the tasks online. The experiment was developed using PsychoPy 3 (Peirce et al., 2019), an open source software package, and Pavlovia was used to host the experiment online. Qualtrics was used to administer the consent form at the beginning of the experiment, the questionnaire at the end of the experiment, and to collect participants' emails if they chose to receive an incentive.

In the advertisement for the study a link to the consent form was provided. The consent form described the background required, the tasks, questionnaire, and compensation (including the list of countries where compensation is not available). After consent, participants were redirected to the experimental portion of the study. The experiment began with a set of instructions asking the participant to determine as quickly and accurately as possible if each program contained a data race and respond by pressing the 'y' or 'n' key on their keyboard. The participants were instructed that the first four stimuli were practice and they would not be included in the results of the study. For each stimulus, participants were given up to 60 seconds to view the stimulus and respond; if they exceeded the time limit exposure to the stimulus ended and they were asked to decide if the stimulus contained a data race or not. The practice set contained two stimuli with a data race and two without, and one stimulus was followed by a question asking participants what cues or program components they used to determine whether or not there was

```c
#include <stdio.h>
#include <omp.h>

int main(int argc, char* argv[]){

        int n = 10;
        int a[n], i, x = 0;

        #pragma omp parallel firstprivate(x)
        {
                #pragma omp for
                for (i = 0; i < n; i++){
                        if(i%2 == 0){
                                x = x + n;
                        }
                        a[i] = x + i;
                }

                #pragma omp for
                for(i = 0; i < n; i++){
                        a[i] = a[i/2] + x;
                }

                printf("%d\n", x);
        }

        return 0;
}
```

(a) Stimuli containing a data race.

```c
#include <stdio.h>
#include <omp.h>

int main(int argc, char* argv[]){

        int n = 10;
        int a[n], i, z = 2;

        #pragma omp parallel
        {
                #pragma omp for
                for(i = 0; i < n; i++){
                        a[i] = z + i;
                }

                #pragma omp sections
                {
                        #pragma omp section
                        a[n-1] = a[n-1] * 3;

                        #pragma omp section
                        z = z + 10;
                }
        }

        printf("%d %d\n", z, a[n-1]);

        return 0;
}
```

(b) Stimuli that does not contain a data race.

Figure 1 – Sample stimuli.

a data race. After the practice set, the instructions were repeated, and the 76 experimental stimuli were presented to the participants in random order. Participants were asked after the data race question to rate their level of confidence in their answer using a visual analogue scale that ranged from "Not Confident" to "Very Confident". For 20 of the stimuli, 10 with a data race and 10 without, participants were asked what cues or program components they used to determine whether or not there was a data race. The data collected from the experiment consisted of the correctness of their response, response time, level of confidence in their response, and the answer describing the program components they used to determine whether or not there was a data race. Correctness of their response refers to whether participants submit a correct answer to the task of identifying the presence of a data race. After completing the experiment portion, participants were redirected to the questionnaire documenting their level of education, programming experience, their perceived level of programming expertise, and age (Feigenspan, Kastner, Liebig, Apel, & Hanenberg, 2012). The questionnaire also solicited feedback on the experiment. Participants were then asked if they would like to receive the e-gift card and, lastly, were redirected to the debriefing.

4.4. Results

For each trial, the accuracy (1 = correct, 0 = incorrect), level of confidence (0 = not confident to 100 = very confident), response time, and whether each trial had a race condition (y) or no race condition (n) were recorded and analyzed for all eight participants in the study using the statistical software program R (R Core Team, 2021). Experiments that were aborted in Pavlovia and consent forms that had no corresponding experimental results from Pavlovia were discarded.

The mean accuracy was .65 (SD = .48) and the mean response time was 24.20 seconds (SD = 17.58). The participants with the greatest mean accuracy (mean = .96 and mean = .79) also had the longest mean response times (mean = 46.57 and mean = 42.16), whereas the participant with the lowest mean accuracy (mean = .46) had the shortest mean response time (mean = 5.65). A one sample t-test was performed to test the null hypothesis that the sample accuracy was equal to chance (mu = .50) with a 95% confidence interval. The results indicated that the accuracy of participants was significantly higher than chance, $t(7) = 2.73$, $p < .05$.

The data were analyzed using a mixed linear model that was fitted with the **lme4** package (Bates, Mäch-

Table 1 – Spearman Correlation Coefficient Between Accuracy, Confidence, and the Measures of Self-Perceived Expertise

	Accuracy
	r
Confidence	.47
Self-estimation of expertise in programming	.45
Self-estimation of expertise in parallel programming	.59
Self-estimation of expertise in programming compared to their peers	.48
Self-estimation of expertise in parallel programming compared to their peers	.37

ler, Bolker, & Walker, 2015) in R. The design used confidence as a continuous predictor, race condition as a repeated measures factor, and accuracy as the dependent variable. Using generalized linear mixed model with the **glmer** procedure from the **lme4** package, we first determined the best fitting model by comparing the fixed slope model and the random slope model. The results of comparing the models show that the random slope model improved fit significantly ($p < .001$). We also compared the random slope model with participant as the random factor to the random slope model with participant and stimuli as random factors. The results of comparing the models show that the random slope model using both participant and stimuli as random factors improved fit significantly ($p < .05$). Likelihood ratio values were then obtained with the **Anova** procedure from the **car** package (Fox & Weisberg, 2019), using the best fitting model with participant and stimuli as random factors with participants treated as random over the intercept, race condition as the random slope component, and accuracy as the dependent variable for confidence and race condition. Results did not show a significant effect of confidence, $LR = 3.35$, $p = .067$ or race condition, $LR = 1.57$, $p = .211$.

Spearman rank correlations were computed to examine the relationship between accuracy, confidence, and the measures of self-perceived expertise. The strength of the correlations was determined using the scale for r values: r = .1 is weak, r = .3 is moderate, r >= .5 is strong (Cohen, 1988). The correlation coefficients are listed in Table 1. There were moderate positive correlations between accuracy and confidence, accuracy and self-estimation of expertise in programming, accuracy and self-estimation of expertise in programming compared to their peers, and accuracy and self-estimation of expertise in parallel programming compared to their peers. There was a strong positive correlation between accuracy and self-estimation of expertise in parallel programming.

The participants with the highest accuracy (mean = .96, .79, .63) provided responses for all 20 stimuli that asked what cues or program components they used to determine whether or not there was a data race. These participants wrote more detailed responses that included specific references to OpenMP directives from the stimuli, and either specific variables, data structures, or programming structures in the stimuli or descriptions of concurrency issues (data sharing, concurrent regions). The participants with lower accuracy did not provide responses to all the questions pertaining to elements relevant to a data race; two of these participants did not provide any responses. The participants with lower accuracy provided shorter, less detailed responses, and some responses were unrelated to the question.

The participant with a mean accuracy of .96 responded within the time limit for 71% of the stimuli. This participant had three incorrect responses, none of which were responses given after the time limit expired. The participant with a mean accuracy of .79 responded within the time limit for 76% of the stimuli. This participant had 16 incorrect responses, and only 5 were given after the time limit expired.

4.5. Discussion

The result of the t-test showing that the accuracy of participants is significantly higher than chance suggests that the level of difficulty of the stimuli is appropriate. The mean response times for all participants were within the 60 second exposure limit implying that the time exposure limit was appropriate.

Although the mean number of lines of code for the stimuli is 20.95 this also includes sequential code (e.g.: declaring and initializing variables, printing), lines containing only an opening or closing brace, and pre-processor directives (i.e.: #include). Given that the task is to determine the presence of data races, it is likely that the high performing participants were able to respond with high accuracy before the time limit expired since the lines of code within parallel regions would be of most importance to the task. Therefore, we hypothesize that participants focused on only select lines of code they felt were critical to the task. For example, in Figure 1a, the lines of code including the directive for the parallel region that contains more than just an opening or closing brace is 10, compared to the total 22 lines of code.

Although the correlations between accuracy, confidence, and self-perceived expertise are not statistically significant it could be attributed to the lack of statistical power given the small sample size. Despite the lack of power, the moderate to strong correlations between accuracy and measures of self-perceived expertise imply that these measures are relevant and worth preserving.

Confidence based assessment which combines accuracy and confidence levels provides four regions of knowledge: uninformed (wrong answer with low confidence), doubt (correct answer with low confidence), misinformed (wrong answer with high confidence), and, mastery (correct answer with high confidence) (Maqsood & Ceravolo, 2018). The moderate positive correlation between confidence and accuracy indicates that higher performing participants had greater confidence in their answers and therefore had higher levels of mastery of the programming language and parallelization directives and were less likely guessing. Lower performing participants tended to have lower confidence which would indicate they lacked knowledge of either the programming language, the parallelization directives, or both, and may have employed more guessing. This indicates that the higher performing participants were from our target population (expert parallel programmers) and that for this target population the level of difficulty and time limit for exposure are appropriate.

4.6. Threats to Validity

In the development of the stimuli, bias may have been introduced due to our background knowledge and familiarity with particular OpenMP directives. To reduce this bias, the majority of the directives used in the stimuli were selected from the common core, considered by Mattson et al. (2018) to be the most prevalent directives used by programmers. The types of errors that we introduced to create data races may have also been biased towards the types of mistakes we most commonly make ourselves. To reduce this bias, we used some of the same types of errors that were found in the DataRaceBench benchmark suite. Bias that may have been introduced by participants would be their prior experiences with data races that may have caused them to look for mistakes they commonly make. Participants may also have more familiarity with some directives than others. Another threat to validity is our lack of control over the experiment environment. Because the study was conducted online, participants may have been in a very distracting environment with other people around them and access to their personal devices.

The small sample size we ended up with was likely the biggest limitation of our work as it greatly reduced statistical power. In an effort to recruit participants, we contacted colleagues at other institutions that shared the advertisement with their colleagues and students who would have the appropriate background. We also emailed the advertisement to past and current students who had studied parallel programming at our institution and to contacts working in the area of high performance computing. A power analysis was performed on the pilot study using the **powerSim** procedure from the **simr** package (P. Green & MacLeod, 2016). The result of the analysis was a power estimate of 46%; this is well below the threshold of 80% power that is considered adequate (P. Green & MacLeod, 2016). Using the power analysis proposed by Brysbaert and Stevens (2018), the pilot study, having 304 observations (38 stimuli x 8 participants), would also not meet the minimum recommendation of 1600 observations. To determine the number of participants for the main study, a simulation using the **powerSim** procedure was performed for 40 participants. The result of the simulation was a power estimate of 98%. Although this exceeds the recommended threshold of 80%, 40 participants would only provide 1520 observations.

We aim to recruit 60 participants (2280 observations) for the main study to ensure that we have adequate power.

5. Conclusion

The results of the pilot study indicate that the time exposure to the stimuli and the level of difficulty of the stimuli are both appropriate and do not need to be adjusted for the main study. The results also indicate that the measures of self-perceived expertise are relevant and should be included in the main study.

We found that for the 20 stimuli that participants were asked what cues or program components they used to determine whether or not there was a data race, the responses varied greatly ranging in level of detail and also in the number responses they provided making it challenging to analyse these data. We speculate that the reason for the variation in responses is due to the open-ended nature of this question. In the main study we will use an approach similar to Burkhardt, Détienne, and Wiedenbeck (2002) and include questions about specific components of the code to determine the type of information that is included in the mental representations formed by participants. We predict that by asking more specific questions we may be able to gain a better understanding of the participants' mental models and that they will be more likely to provide answers to more direct questions. Therefore, in the main study, 12 of these 20 stimuli, six with a data race and six without, will be given more specific questions regarding the data structures in the programs. For eight of the stimuli, four with a data race and four without, participants will still be asked what cues or program components they used to determine whether or not there was a data race.

In adding questions for the main study, we also considered that, in addition to the program and situation model, an execution model has been proposed that includes the behaviour of data structures (Aubanel, 2020). The behaviour of data structures is considered an important part of parallel program comprehension. Specifically, how the data structures are accessed and changed by one or more threads and the relationship between data structures, is particularly important for determining if a data race exists. The importance of data structures is also evident by the responses in the pilot study when asked what program components were used to determine if there was a data race or not. Of the 60 responses from the highest performing participants, 11 responses referred to data structures, either by name or to the elements within a data structure. We propose that although the data structure behavior is important, the contents of the data structures and the size of the data structures will not necessarily be part of the participants' mental models as those details are not critical to the task. The two types of questions that will be asked of participants are questions related to the contents and size of data structures, and the other is related to accessing and changing of the contents of the data structures and relationships between data structures. Six stimuli, three with a data race and three without, will be selected for each type of question.

The main study has been preregistered with Open Science Framework (OSF) (Bidlake, 2022) and the stimuli, data, and R code will be made available upon completion of the project (pilot and main study).

6. References

Abbes, M., Khomh, F., Guéhéneuc, Y.-G., & Antoniol, G. (2011, Mar). An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension. In *2011 15th european conference on software maintenance and reengineering* (p. 181–190). doi: 10.1109/CSMR.2011.24

Arab, M. (1992, 2). Enhancing program comprehension: Formatting and documenting. *ACM SIGPLAN Notices*, *27*(2), 37–46. doi: 10.1145/130973.130975

Aubanel, E. (2020). Parallel program comprehension. In *31st Annual Workshop of the Psychology of Programming Interest Group (PPIG 2020).*

Barfield, W. (1997, 12). Skilled performance on software as a function of domain expertise and program organization. *Perceptual and Motor Skills*, *85*(3, Pt 2), 1471–1480. doi: 10.2466/pms.1997.85.3f.1471

Bates, D., Mächler, M., Bolker, B., & Walker, S. (2015). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, *67*(1), 1–48. doi: 10.18637/jss.v067.i01

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*(6), 649–660. doi: 10.1016/S0020-7373(87)80058-5

Bavota, G., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D., & De Lucia, A. (2013, May). An empirical study on the developers' perception of software coupling. In *2013 35th international conference on software engineering (icse)* (p. 692–701). doi: 10.1109/ICSE.2013.6606615

Beniamini, G., Gingichashvili, S., Orbach, A. K., & Feitelson, D. G. (2017, May). Meaningful identifier names: The case of single-letter variables. In *2017 ieee/acm 25th international conference on program comprehension (icpc)* (p. 45–54). doi: 10.1109/ICPC.2017.18

Bergantz, D., & Hassell, J. (1991). Information relationships in prolog programs: How do programmers comprehend functionality?. *International Journal of Man-Machine Studies*, *35*(3), 313–328. doi: 10.1016/S0020-7373(05)80131-2

Bhatia, S., & Malhotra, J. (2014, Aug). A survey on impact of lines of code on software complexity. In *2014 International Conference on Advances in Engineering & Technology Research (ICAETR - 2014)* (p. 1–4). doi: 10.1109/ICAETR.2014.7012875

Bidlake, L. (2022, Feb). *Validation of stimuli for studying mental representations formed by parallel programmers during parallel program comprehension.* OSF. Retrieved from `https://osf.io/fcnyx/` doi: 10.17605/OSF.IO/FCNYX

Bidlake, L., Aubanel, E., & Voyer, D. (2020, July). Systematic literature review of empirical studies on mental representations of programs. *Journal of Systems and Software*, *165*, 110565. doi: 10.1016/j.jss.2020.110565

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* (pp. 567–576). doi: 10.1145/1240624.1240715

Brysbaert, M., & Stevens, M. (2018). Power analysis and effect size in mixed effects models: A tutorial. *Journal of Cognition*, *1*(1). doi: 10.5334/joc.10

Burkhardt, J.-M., Détienne, F., & Wiedenbeck, S. (2002). Object-oriented program comprehension: Effect of expertise, task and phase. *Empirical Software Engineering*, *7*(22), 115–156.

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed ed.). Hillsdale, N.J.: L. Erlbaum Associates. Retrieved from `http://www.gbv.de/dms/bowker/toc/9780805802832.pdf`

Davies, S. (1990). The nature and development of programming plans. *International Journal of Man-Machine Studies*, *32*(4), 461–481. doi: 10.1016/S0020-7373(05)80143-9

Détienne, F. (2001). *Software design-cognitive aspect*. Springer Science & Business Media.

Feigenspan, J., Kastner, C., Liebig, J., Apel, S., & Hanenberg, S. (2012, Jun). Measuring programming experience. In *20th IEEE International Conference on Program Comprehension (ICPC)* (pp. 73–82). IEEE. doi: 10.1109/ICPC.2012.6240511

Fox, J., & Weisberg, S. (2019). *An R companion to applied regression* (Third ed.). Thousand Oaks CA: Sage. Retrieved from `https://socialsciences.mcmaster.ca/jfox/Books/Companion/`

Furman, S. M. (1998). *Improving software comprehension* (Unpublished doctoral dissertation). ProQuest Information & Learning.

Gilmore, D. J., & Green, T. R. G. (1988). Programming plans and programming expertise. *The Quarterly Journal of Experimental Psychology A: Human Experimental Psychology*, *40*(3-A), 423–442. doi: 10.1080/02724988843000005

Green, P., & MacLeod, C. J. (2016). simr: an r package for power analysis of generalised linear mixed models by simulation. *Methods in Ecology and Evolution*, *7*(4), 493–498. Retrieved from `https://CRAN.R-project.org/package=simr` doi: 10.1111/2041-210X.12504

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 (pp. 443–460). New York, NY, USA: Cambridge University Press.

Liao, C., Lin, P.-H., Asplund, J., Schordan, M., & Karlin, I. (2017). Dataracebench: A benchmark suite for systematic evaluation of data race detection tools. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 11:1–11:14). ACM. doi: 10.1145/3126908.3126958

Maqsood, R., & Ceravolo, P. (2018, Jul). Modeling behavioral dynamics in confidence-based assessment. In *2018 IEEE 18th International Conference on Advanced Learning Technologies (ICALT)* (p. 452—454). doi: 10.1109/ICALT.2018.00112

Mattson, T., Koniges, A., He, Y. H., & Chapman, B. (2018). *The OpenMP Common Core: A hands on exploration.* SC.

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

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

Peirce, J., Gray, J. R., Simpson, S., MacAskill, M., Höchenberger, R., Sogo, H., ... Lindeløv, J. K. (2019, Feb). Psychopy2: Experiments in behavior made easy. *Behavior Research Methods*, *51*(1), 195–203. doi: 10.3758/s13428-018-01193-y

Pennington, N. (1986). *Stimulus structures and mental representations in expert comprehension of computer programs.*

Pennington, N. (1987a). Empirical studies of programmers: Second workshop. In G. M. Olson, S. Sheppard, & E. Soloway (Eds.), (pp. 100–113). Norwood, NJ, USA: Ablex Publishing Corp.

Pennington, N. (1987b). Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, *19*(3). doi: 10.1016/0010-0285(87)90007-7

R Core Team. (2021). R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria. Retrieved from `https://www.R-project.org/`

Ramalingam, V., LaBelle, D., & Wiedenbeck, S. (2004, Jun). Self-efficacy and mental models in learning to program. In *Proceedings of the 9th annual SIGCSE conference on Innovation and Technology in Computer Science Education* (p. 171–175). New York, NY, USA: Association for Computing Machinery. Retrieved from `http://doi.org/10.1145/1007996.1008042` doi: 10.1145/1007996.1008042

Shargabi, A. A., Aljunid, S. A., Annamalai, M., & Zin, A. M. (2020, Jul). Performing tasks can improve program comprehension mental model of novice developers: An empirical approach. In *Proceedings of the 28th International Conference on Program Comprehension* (p. 263–273). New York, NY, USA: Association for Computing Machinery. Retrieved from `http://doi.org/10.1145/3387904.3389277` doi: 10.1145/3387904.3389277

Soloway, E., & Ehrlich, K. (1984, Sep). Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, *SE-10*(55), 595–609. doi: 10.1109/TSE.1984.5010283

Teasley, B. (1994). The effects of naming style and expertise on program comprehension. *International Journal of Human - Computer Studies*, *40*(5), 757–770. doi: 10.1006/ijhc.1994.1036

Tubaishat, A. (2001). A knowledge base for program debugging. In *AICCSA '01 Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications* (Vol. 2001-January, pp. 321–327). doi: 10.1109/AICCSA.2001.934005

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

Wiedenbeck, S., & Ramalingam, V. (1999, 07). Novice comprehension of small programs written in the procedural and object-oriented styles. *International Journal of Human-Computer Studies*, *51*(1), 71–87. doi: 10.1006/ijhc.1999.0269