Introductory Research of Human Factors In Concurrent Programming

by

Zhangliang Ma

Bachelor of Cognitive Science, Rensselaer Polytechnic Institute, 2020

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s):	Dr. Eric Aubanel, Faculty of Computer Science
Examining Board:	Dr. Luigi Benedicenti, Faculty of Computer Science, Dean
	Dr. Daniel Rea, Faculty of Computer Science,
	Dr. Veronica Whitford, Department of Psychology, Faculty of Arts
External Examiner:	N/A

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

August 2022

Table of Contents

Abstract

Researchers have found that concurrent programming is difficult to learn and hard to master, and people who have little experience in concurrency tend to overlook some key features of concurrent programs, resulting in unexpected program behavior. To understand how people understand concurrent programming, we conducted a study focusing on the mental representation of concurrency. This study is dedicated to finding and understanding the relationship between the three types of mental models: program model (the code structure), situation model (the meaning and purpose of the code), and execution model (the machine-level execution pattern) involved in the understanding of concurrent programs. We designed a questionnaire concerning these three models with different questions categorized into either of these models. This questionnaire measures programmers' understanding of a concurrent Java program. The experiment divided the participants into two groups and assigned a different task to each group. The tasks are named either "read-to-recall" (participants need to describe the purpose of the program) or "read-to-do" (participants need to find bugs). Participants were instructed to read the code while describing the purpose or bugs of the code first, and were then asked to answer a series of questions immediately after without the presence of the code. The results show that no significant difference exists between these two types of tasks (for the purpose describing or bug finding) when people try to understand the program. On the other hand, we found that there exist a few

differences in programmers' understanding of the three types of mental models, as seen by how accurately participants answered the questions.

Chapter 1

Introduction

1.1 Current Research Problem

Concurrent programming is a type of programming that supports simultaneous access to memory or shared data by multiple threads. The use of multi-threading enables the data to be shared among threads, so that system accesses can happen concurrently and simultaneously. However, concurrent programming can also be very challenging, since one must ensure the runtime correctness of a program while the threads can access the shared data in various orders [1]. It requires an understanding of concurrent features, since the machine cannot control the execution order of threads to ensure a correct result [1].

Programmers need to build fully functional mental models if they want to write useful and correct programs. Conventional serial programming requires programmers to build a mental model with the correct syntax, code structures, and control. This model is called the program model [1]. Then, the mental model that helps people understand the meaning of the code, such as the goal of a program and relationships between different objects (classes), is called the situation model [1]. By contrast, comprehension of concurrent programs not only requires

syntax-level understanding, but also requires a mental model of machine execution (execution model) hidden from the written words on the screen, because the hardware level execution is not reflected in the program syntax [1]. To understand how these mental models are constructed we require studies of the cognitive aspects of concurrent programming, yet there is a lack of research on mental models of concurrent program comprehension [7]. Therefore, a study of concurrent programming mental models is crucial for the analysis and understanding of concurrent programming itself.

Even though studies have been done concerning the understanding of object-oriented programming, no study has never done on concurrent programming [1][21][9].

1.2 Outline of Our Study

The goal of this study is to understand the mental model structure of programmers, and if any code comprehension task can affect this structure. The measurement of the mental model was done with comprehension questions we provided to the participants. For the participants to understand the code, we asked them to read the code while they were asked to finish a comprehension task in a limited amount of time. Some participants were asked to describe the purpose of the program, and some other participants were asked to look at concurrency bugs within the code. At the end of the experiment, we measured how correctly participants answered the questions, to understand the mental model structure built by the participants. We then compared the mental model structure (correctly answered questions) between the subjects, to find if the tasks assigned to them made any difference.

Chapter 2

Background

To understand why it is difficult to code correctly using concurrency, we need to understand concurrency first, and the potential issues in code that is seemingly correct.

2.1 Pitfalls of Race Condition and Data Dependency

To understand exactly why programmers fail at writing fast and correct concurrent programs it is necessary to understand how people approach the code structure first. Consider the following code example provided by Aubanel: [1]

```
double a[len][len];
\\ ...
#pragma omp parallel for private(j)
for (i = 0; i < len - 1; i += 1) {
    for (j = 0; j < len ; j += 1) {
        a[i][j] += a[i + 1][j];
    }
```

}

If we ignore the line with the *#pragma* notation (the notation that tells the compiler to assign the loop iterations to many threads) and only read the rest of the code, we can easily find a nested loop that modifies the value of an array with some other elements in the same array. Usually programmers build this model automatically, if they are familiar with this type of language [1]. If we trace the situation model of this program, we can find the data flow by which this program updates the array cell with the value of the same cell in the next row. Indeed, this is also the goal of this program [1].

However, the introduction of the *#pragma* notation changes everything. This line of code tells the compiler to create machine code that runs concurrently. It brings us an implicit meaning that the array a is shared among threads but both the loop variables i and j are not [1]. If we analyze the data flow of this program we may find that when one thread is writing to *a[i][j]*, it needs to read the data from the cell *a[i+1][j]*. However, at the same time another thread has already written, is still writing, or has not accessed the data in the cell *a[i+1][j]* [1]. The result is dependent on chance, and such behavior is called a "data race" [1].

The challenge for the programmer is how to understand the implicit information correctly and precisely when the *#pragma* appears, since the line with such information does not convey the explanation above directly.

2.2 Data Races and Data Dependencies

As we illustrated in the code above, there exists a data race and some data entries are dependent on others. This is due to the fact that the timing of threads is controlled by the computer system, but not by the programmers themselves. Consider the following code:

```
int i, a = 1;
#pragma omp parallel for
for (i = 0; i < 2; i += 1) {
    a = a + 1;
}
```

All this program does is to add 1 to value a twice in a row. The *for* loop calls $a = a + 1$ when it first encounters this line. Then it calls it again to add another 1 to this variable. To make it run faster, we can try to add the value 1 to a twice simultaneously using concurrency. However, things get complicated here.

At the machine level, even the seemingly most basic operations such as $a = a + 1$ are composed of multiple steps defined by the natural property of assembly languages and machine instructions. The machine-level execution of this program statement can be summarized as follows:

1. **Step 1:** The CPU reads the value of the variable a from the RAM into one of the registers
2. **Step 2:** The Arithmetic Logic Unit (ALU) adds 1 to the value held in that register
3. **Step 3:** The CPU writes the calculated value from the register back to the RAM

If we simply perform the 3-step operation in parallel without synchronization, the thread operation sequence is unpredictable and may lead to an unpredictable result [17]. If we allow two threads to call $a = a + 1$, either of the following operation patterns may happen. We use underlines to represent different threads:

1. **Thread 0**: read the value from a (a = 1) to a register *$t0*
2. **Thread 0**: add 1 to the value held by *$t0* (*$t0* = 2)

3. **Thread 0**: write the value from *$t0* to the memory location variable *a* holds (a = 2)

4. **Thread 1**: read the value from *a* (a = 2) to another register *$t1*

5. **Thread 1**: add 1 to the value held by *$t1* (*$t1* = 3)

6. **Thread 1**: write the value from *$t1* to the memory location variable *a* holds (a = 3)

or:

1. **Thread 0**: read the value from *a* (a = 1) to a register *$t0*

2. **Thread 1**: read the value from *a* (a = 1) to another register *$t1*

3. **Thread 0**: add 1 to the value held by *$t0* (*$t0* = 2)

4. **Thread 1**: add 1 to the value held by *$t1* (*$t1* = 2)

5. **Thread 0**: write the value from *$t0* to the memory location variable *a* holds (a = 2)

6. **Thread 1**: write the value from *$t1* to the memory location variable *a* holds (a = 2)

These are just some basic examples. In reality, any overlapping pattern may occur.

As we find in step 6 of each calculation, the variable *a* has a value of either 2 or 3, depending on how the threads interleave between one and another [17]. This will undermine the correctness of the program and may lead to a catastrophic result.

However, if we carefully analyze the *for* loop, we can find that during the second call of *a = a + 1*, the result of *a* read to the CPU is 2, which is dependent

on the first call of the same line. Thus, we also find a data dependency problem in this code here, as it prevents us from parallelizing the *for* loop without synchronization. Several solutions to this problem exist, but they are not our study's concern.

2.3 Previous Studies

Programming languages are a subset of scripts, which can be understood as determined actions that lead to a specific situation [25]. These scripts typically require knowledge from our understanding of the world around us, and people have been studying our reasoning process of scripts for decades [25]. Early in 1977 Roger C. Schank and Robert P. Abelson published a book summarizing how our languages and words form a "story" with transition, goals, plans, and themes [25]. The two researchers concluded that for a proper "story" text to be logically acceptable, it must have at least one plan, one goal, and one theme [25]. Among these three components, the plan is the general information that connects the events, the goal is the materialization of our expectation of the final outcome of a series of events, while the theme is the collection of meta-parameters that monitor the flow of the story, and we often refer to the theme as "situation" [25]. These three aspects of a story text exist in every meaningful scripting language, and they form the fundamental parts of our mental model of human understanding of such text languages. However, the role of the mental model in concurrent programming has not yet been extensively researched.

Even though the analysis of concurrent programming mental models has not been conducted before, there have been many cognitive studies concerning serial programming [7]. In the 1980s, Pennington conducted a study of programming comprehension involving the control flows and goal hierarchy among different

parts of the code based on the theories of Schank and Abelson [21]. Pennington's central focus was whether the control flow or the goal hierarchy relations make a major part of a programmer's mental representation of any given program [21]. The results of her study showed that the control flows win against the goal hierarchy while there are many different abstraction levels hidden in the context of a piece of code [21]. Also, to understand a piece of code, the programmer must have understood various types of knowledge in order to fully understand its true purpose and the method to achieve the final goal. Also, Pennington's work is the first introduction of program and situation models.

To understand the situation model more precisely, let us examine the following code section:

```
1       int len = 10;
2       double a[len][len];
3
4       int i, j;
5       for (i = 0; i < len; i += 1) {
6               for (j = 0; j < len ; j += 1) {
7                       a[i][j] = i*j;
8               }
9       }
10
11      double sum = 0;
12      int i, j;
13      for (i = 0; i < len; i += 1) {
14              for (j = 0; j < len ; j += 1) {
15                      sum += a[i][j];
16              }
```

```
17      }
```

This code section gives us the following information:

1. lines 1-2: Defines a 2-D array with the dimensions of 10*10. All the cells host double floating-point numbers

2. lines 4-9: A nested *for* loop traverses through all the cells, assigning each cell with the product of the current value of i and j

3. lines 11: Defines a double floating-point with the name of "sum", which implies that we will use this variable to calculate a value in which different items are added

4. lines 12-17: A nested *for* loop traverses through all the cells, adding each of the cells value to the *sum*

If we summarize each piece of code sections above into a flow of calculation, we can find that this program defines a 2-D array, assigns each cell an initial value, and sums up the total value of this array. This is also the goal of the program in our case. More complex code will involve the relations between objects and a more advanced flow of data, and their goals might also change, depending on how we would like to use the code. However, the fundamentals remain the same regardless of the complexity of the code: meaningful texts of scripting languages must contain the plan, the goal, and the theme, and readable code must contain such information that defines the behavior of the code [25].

In 2002, Burkhardt et al. conducted a study concerning Object-Oriented (OO) language program comprehension[9]. In Burkhardt et al.'s paper, the experiment is a between-subject design divided into two phases. The first phase consists of program studying which asks the participants to read the code and understand the design philosophy while the second phase asks the two groups of participants

to either write documentation or reuse the code for other programs, and then both groups needed to answer the comprehension questions concerning the OO features [9]. Their results show that both novices and experts are capable of understanding the text-based code structure (program model) yet the experts constructed a significantly better situation model in terms of more comments or more reuses [9]. These results should also apply to our study of concurrent programming, in that experienced programmers can demonstrate a better situation model performance in a more proficient debugging and a deeper understanding in reading code. The appendix of this paper provides a questionnaire with questions concerning all aspects, such as the "purpose of this program" and "relationship among classes" [9]. These materials offer us a model of a program understanding experiment and the methods and materials to prepare for it.

In 2013, Lin and Dig used an Eclipse IDE extension named CTADetector and found that even experienced programmers and mature projects can fall into Java concurrency pitfalls [18]. Their study is based on the package "java.util.concurrent. ConcurrentHashMap", and the CTADetector is able to find the code snippets that may cause the program to fail on the concurrency level. The names of these Java methods suggest that they have classified several types of concurrent misuses and named them "idioms". These idioms consist of "*Put-if-absent*" (based on the method ConcurrentHashMap::putIfAbsent()), "*Get*" (based on the method ConcurrentHashMap::get()), "'*Remove*" (based on the method ConcurrentHashMap::remove()), "*Replace*" (based on the method ConcurrentHashMap:: replace()), "*Queue*" (based on the method ConcurrentLinkedQueue::poll()), "*Lazy Initialization*" (initialize a new instance without proper checks), and some other misuses of lists [18].

To understand what these idioms refer to, Lin and Dig presented them in context using code snippets. We can demonstrate one of the idioms here. Consider

the “”*Put-if-absent*” idiom from their work: [18]

```
import java.util.concurrent.ConcurrentHashMap;

ConcurrentHashMap<Key, Value> m =
        new ConcurrentHashMap<Key, Value>();

Key key;
Value v = m.get(key);
if(v == null){
        v = calc();
        map.putIfAbsent(key, v);
}
return v;
```

What this code snippet does is check whether the key-value pair exists in the map. If not, then it calculates a value and map it to the ConcurrentHashMap with the given key. The pitfall here is that when a thread, T0, calls this idiom first and tries to find the key-value pair but none exists, it will take some time to calculate the value of v. However, when another thread, T1, calls the same idiom at the same time, but finishes the calculating of its corresponding value v faster than that of the thread T0, it will write the key-value pair with its own value first. Then, immediately after this, thread T0 finishes the calculation and tries to put its value under the same key, but it fails due to the property of the *putIfAbsent()* defined by the JDK tools. The returned v of thread T0 does not exist in the map.

Yu Lin and Danny Dig showed that these idiom traps constitute 78%, 11%, 3%, 1%, 1%, 1%, and 5%, respectively, of all the concurrent idiom errors mentioned above, in well-established existing program releases [18]. It thus shows how easily that even the expert programmers make mistakes while implementing some of the

basic modules in concurrent programming.

2.4 Current Research Progress

A systematic literature review by Bidlake, Aubanel and Voyer shows that few mental model studies of programming have been conducted since 2005 in the field of empirical studies on program comprehension [7]. The literature review shows that the major focus of comprehension studies since 1975 has been on program studying, modification and maintenance (74 of 98 studies are analyses of these 3 task types). The remaining 24 studies cover other tasks such as debugging, reusing, classifying, documentation, writing/reconstructing, hand execution, enhancement, and recopying, demonstrating a huge skew in the distribution of tasks [7]. The reason for the recent lack of research is that researchers shifted their focus to comparisons of programmers with different backgrounds in various programming environments and languages [7].

Mental models of concurrent programming have not yet been studied even though many concurrent programs have already been deployed [7][23]. Concurrent systems have suffered from a lack of appropriate metrics and tools, and inadequate measurement leads to the lack of a proper guide to developing these tools, since such measurement requires an extra level of human-factor understanding [1][7][23]. Concurrent programming is difficult since the code blocks can be correct at a syntax level, while on the machine level they may contain bugs such as race conditions and data dependency problems as demonstrated above [1][2]. Therefore, studies concerning models of concurrent programming need to be conducted in order to test the cognitive aspects of the programmers, and possibly build some tools or languages to foster the mental model building process to help develop tools to help programmers build correct concurrent programs.

In 2020, Aubanel proposed an extension to the existing program/situation mental models in his paper "Parallel Programming Comprehension", and he proposed a new model, the execution model, which separates the data flow from the existing situation model [1]. This execution model shows the behavior of data structures and the data flow of the machine. More importantly, it also represents the decomposition and communication of shared data between threads [1]. With the aid of the program model, the situation model, and the execution model, we can develop a proper measurement tool to fully analyze the human factors of concurrent programming.

2.5 Statistical Models and Power Analysis

Many psychological studies use the linear-mixed effect model, as it offers a more comprehensive analysis compared to the General Linear Model (general multivariate regression model), which underlies simpler approaches such as ANOVA and t-test [4]. Previously, however, many studies had utilized the ANOVA or regression approach to handle categorical within-unit predictors with repeated-measures designs, but ANOVA by itself is inadequate in handling cases where the same subjects were presented with the same stimuli [8].

Brysbaert and Stevens also mentioned that many psychological studies lack sufficient statistical power due to various types of miscalculations [20]. To compensate for the loss of the statistical power, they recommend two different programs and applets that help break down this issue [20]. However, many of the variables required by these power calculators require parameters that are not attainable during the early stages of the study, making these power tools more suitable for calculating power score on the existing studies than for estimating the required number of participants for ongoing studies. To make the estimation

easier, Brysbaert and Stevens offered a "golden rule of thumb" that the number of participants multiplied by the number of stimuli presented to each participant should be above 1,600 ("40 participants * 40 stimuli per participant" or "80 participants * 20 stimuli per participant", depending on the designs of the studies) [20].

Some other materials offer us ways to estimate the number of participants based on a certain level of error (alpha value) and power. Barcikowski and Robey's work in 1985 presented several tables showing the number of participants with a given effect size (Cohen's d), various alpha values and a fixed power score of 0.80 [3]. As most of the studies use the alpha value of 0.05, the only parameter required is the effect size, which is easily obtainable during the pilot study.

Chapter 3

Method

We asked the participants to either describe the purpose of, or look for concurrent bugs in, our program. The hypothesis was that there existed differences in people's constructions of differernt types of concurrent mental models (program model, situation model, and execution model), and we expected that debugging (bug description) would be more likely to lead to people forming the execution model compared to documentation writing.

3.1 First Pilot Study

We started our work with a program and questionnaire that included many features, especially the concurrency "idioms" from Lin and Dig's work [18]. Their idioms were widely used in many well established projects such as *Annsor* and *Apache Cassandra* [18]. We hoped that through the indroduction of these idioms we would build an experiment scenario which is very close to the professional developer's environment. Therefore, we chose four idioms (*Get*, *Remove*, *Replace*, and *PutIfAbsent*) and constructed serveral related classes to support them.

The code from this version featured a demo of a *Library* which allows people to borrow and return books at the same time while it registers some information

about the books and the person who borrows the book. In addition, the "librarian" can also add new books or remove existing books from the library. During the pilot study, we noticed that this version was probably too complex and required a significant effort to understand. This made it difficult for the participants to locate the bug we expected them to identify. As a result, the experiment lasted 1 hour, and the result was that only 6.4% of the participants (3 completed, 44 aborted) finished the entire experiment during the one-month period of the pilot study. Furthermore, the *Library* structure relies on the use of *java.util.concurrent* package, which further limited our access to our participants.

The estimated time of completion of the first version of the experiment was expected to be 1 hour. However, since the participants needed to answer the mental model questionnaire without the presense of the code, they had to remember a considerable amount of the features as there were 40 questions. As a result, only 3 out of 47 actively running sessions were complete, which indicated we had to shorten the code and questionnaires to find more participants.

The code and questionnaire used in this version are attached to Appendix F, G, H. To resolve the problems we encountered in this version, we designed a new version of the study with a much easier and straightforward code and questionnaire.

3.2 Outline of the Study

To understand the relationship between concurrent programming and both the situation model and execution model, our study would provide the participants with a concurrent Java program. The code can be found in Appendix A. Hints such as the general goal of the program, relationship between objects and classes, data flow, etc. were not presented to participants. The participants were also

provided with a questionnaire concerning both concurrent and non-concurrent aspects of the code.

We adapted our experiment design from Burkhardt's work [9]. The participants were divided into two groups, with each group either describing the purpose or looking for bugs by reading the same code. Then, they answered the same questionnaire after their respective task. The questionnaire contained program model, situation model, and execution model questions assigned to participants in the document taker and debugger group. The questions were all Yes/No questions, such as: "Does the program define the 'Account' class? (Y/N)" (Appendix B).

At the end of the experiment, the participants completed a demographic questionnaire (Appendix D). This questionnaire provided a self-assessment so that we could find the correlation between the self-assessment and the performance result to inform our conclusions. The correlation analysis consists of a cross-analysis between the questions from the mental model questionnaire categorized by the three types of mental models, and the self- and peer-assessment score each participant reported in the demographic questionnaire.

The experiment of this study was a between-subject design with two conditions: 1) read-to-recall task; and 2) read-to-do task. Both groups used the same code as the stimulus, but the task presented to each group was different.

3.2.1 Read-to-Recall

A Read-to-Recall task means that the participants needed to read a piece of code. In this task, the participants needed to write their own understanding of the purpose of the given program, and then answer a questionnaire concerning their understanding of the code. However, further information was not provided because the participants needed to read and understand the concurrency part of the program independently. After finishing this task within 15 minutes, the

participants completed a questionnaire concerning all the three models (program model, situation model, and execution model).

3.2.2 Read-to-Do

A Read-to-Do task means that participants approach a piece of code with the priority of bug finding instead of understanding the code first [9]. In this Read-to-Do task, participants reported whether they found a concurrent bug or not, based on the same code provided to the Read-to-Recall group in the same time duration as that of the Read-to-Recall task (15 minutes). Yet the code itself did not contain any bugs. The bug report was not evaluated in this study, but they could provide important information for future studies. After they finished this task, the participants completed the same questionnaire as mentioned in the Read-to-Recall group.

3.3 Code

The code we use in this study can be found in Appendix A.

We adapted the code from Venkatesh Prasad Ranganath and John Hatcliff's work "Slicing Concurrent Java Programs using Indus and Kaveri" [22]. Then, we changed the class titles so that participants with different cultural and economic background were not offended, since the terms and notions used in this piece of code is relatively archaic compared to modern values.

The code we use contained 4 different classes, and each of them provided different features that programmers may frequently encounter in their actual coding tasks:

1. **Account:** This class holds value of the information that other objects can gain access to. Two of the concurrent methods lock the thread accesses

when the *amount* is less than 0.

2. **Employer:** This class adds value to the field *amount* of the *Account* class. Implementation of Java *Runnable* enables multiple instances of this class to run concurrently.

3. **Employee:** This class removes the value from the field *amount* of the *Account* class. Implementation to Java *Runnable* enables multiple instances of this class to run concurrently.

4. **Business:** Hold the main program. It also shows how many threads are assigned to different instances of objects.

Although this is a short program, it offers various relationships among threads. The *Account* class not only offers methods that use thread-locks, but also put a thread in a waiting state when its field *amount* was below a threshold. This could happen on multiple instances of the *Employee* so that all of them were blocked and could not gain access to the field *amount*, until the *Employer* successfully added the money, and called the built-in *notifyAll()* to unblock all the threads assigned to the *Employee*.

In the *Employee* class, each time an object instance of this class is running, it moves the "money" into its private *Account*, which is not accessible by other objects or other threads.

3.4 Questionnaire

We adapted the questionnaire format from Appendix 2 of Burkhardt's work, but it was not necessary to randomize the order of the questions in our study [9]. The reason for this was that we only measured the understanding of the concurrent code but not the reaction time or the memory load. It was also required that

to understand the concurrent part of the code, one must first understand the fundamentals, such as data flow and object relationships. However, it did not render the program model or the situation model questions useless as we need to use them as baselines to compare the accuracy (percentage of the questions answered correctly in its field). Burkhardt had already demonstrated that the order of the questions did not have a significant impact on the overall level of understanding of the program [9].

Our questionnaire had a total of 16 questions, with 5 questions dedicated to the program model, 6 questions dedicated to the situation model, and another 5 questions evaluating the understanding of execution model. Questions in this questionnaire were all polar questions (Yes/No) and the distribution of the "Yes" and the "No" were randomly assigned to different questions, so that the participants could not find a specific pattern and undermine the result of our experiment. During the experiment, the participants saw the questions according to the order listed in Appendix B, but they were not informed if the questions were asking about their understanding of either the program model, situation model, or execution model.

3.5 Demographic Questionnaire

After the participants finished the mental model questionnaire, we also provided them a demographic questionnaire. This questionnaire asked the participants for their age, experience level and self-rating of their concurrent programming skill. We hoped to find a correlation between the mental model questionnaire performance (number of correctly answered questions) and the level of expertise in concurrent programming. This expertise can be revealed by the types of mental model, the language, and the task assigned to the participants because correlation

analysis would be run regarding all the three categories.

3.6 Experiment Platform

The experiment was hosted on pavlovia.org with the help of an experiment designing tool named as PsychoPy [26]. PsychoPy offered different tools and options to create the backbone of the experiment. After the draft of the experiment was finished we uploaded the experiment and files to pavlovia.org to conduct online studies. During this phase the Python code generated by PsychoPy was automatically converted into a Javascript code to be accepted by the web browser.

3.7 Participants

Participants were chosen from online programming forums. We needed to ensure that the participants at least understood concurrency and the fundamentals of Java such as objects, method calling, and Java documentation, since they needed to read the code in order to understand how the concurrency was represented in the syntax. It is easy to tell if a person has experience in their answers to some questions in the demographic questionnaire, as well as from their comments. Understanding of concurrency was required, but it would be better to have various levels of expertise so that it was possible to find the distribution of the mapping between the questionnaire performance and the presence/strength of the execution model.

We found the participants from social media and online platforms, mainly Reddit channel *r/SampleSize*, *r/ProgrammingBuddies*, study hosting websites such as *SurveyCircle*, and direct reference by those who had already participated and finished the experiment. To do so we made an advertisement that stated the purpose of the study, provided the participants with the basic information, and required

the eligibility. This advertisement can be found in Appendix C.

We also translated the experiment, including the consent form, task descriptions, and questionnaires, into Chinese, in the hope of increasing the diversity of the participants.

3.8 Study Progress

Before we launched our experiment, we conducted a pilot study to eliminate some design errors to ensure the validity of data.

For this pilot study, we recruited people from those we know and others from social media with an early release of its online version so that it was relatively fast to identify the potential issues and make minor corrections before conducting the main experiment. People who participated in the pilot study were given the choice of being awarded with a CAD $10 gift card. During this pilot study phase, we collected the data and analyzed the reaction of the participants, which guided the formal release version of our experiment.

After the participants finished the pilot study, they were not given the correct answers to the questionnaire and the debugging performance report because that information could serve as spoilers for future participants.

During the main study, the participants were recruited via emails, coding forums, and social media. The material we used to hire them was the same advertisement posted on the afore mentioned social media channels (Appendix C).

3.9 Testing and Evaluations

Participants' accuracy was directly related to the number of correct answers on the mental model questionnaire. The data were analysed using a linear-mixed

effect model to eliminate the errors induced by the repeated measure used in our study [13][8][4]. The random factor only existed among the participants because we delivered the same stimuli for both groups, and we had a repeated measurement in our questionnaire because the questions might affect each other [4]. Our study contained a questionnaire that not only exposed repeated stimuli to the participants but also yielded categorical results, thus making t-test or ANOVA inaccurate to evaluate the data [4].

The combination of the program, situation, and execution model questions was to find whether the participants build the situation model or execution model and whether recalling or doing is more effective in building these models. The distribution of correct answers should reflect the model that the participants built during concurrent program comprehension.

3.10 Pilot Study

As the study progressed we launched two different versions of the pilot study. We noticed that the first pilot study revealed several issues, including that the program and questionnaire were too long, which participants complained to be too challenging. Therefore, we refitted this experiment with a shorter, bug-free, and peer-reviewed program and designed the questions based on its structure, meaning, and execution pattern.

This study experiment was registered and approved by the ethics department of UNB with the approval number of 2022-067, and went through major revisions to change the code and questionnaire.

This study was posted online, and we hoped to find participants using social media. We awarded the participants who voluntarily left their email address with e-giftcards from Amazon for the English-speaking subjects and JingDong (the

Chinese equivalent of Amazon, as Amazon is not very popular in China) for the Chinese-speaking subjects.

We released the code and the mental model questionnaire in Appendices A and B. With the new code and questionnaire, we expected that the experiment would only last approximately 30 minutes, which was only half the duration compared to the previous version. It also helped reduce the mental load of the participants as they only needed to answer fewer questions, since the smaller number of questions also meant fewer features to be remembered. At the same time, we still provided the same giftcard award to the participants, which meant that their time was made more valuable.

The pilot phase lasted one month, from mid-July 2022 to mid-August 2022. During the pilot study, we allowed the participants to leave us their feedback so that we were able to find potential issues from the study. Overall, we found no serious issues, perhaps since the experiment was designed to be shorter, as the code used in this version is a peer-reviewed bug-free program. The only feedback we received was that one person complained that we needed to show them the code alongside the questionnaire. However, doing so would violate our original purpose of the mental model study, as participants needed to remember the features of the code within a limited time interval (15 minutes). Therefore, we added a related notification to the informed consent form, telling particants that the code would not be visible while they were answering questions.

We collected 11 fully finished responses in this phase. With the help of these data, we calculated the effect size (Cohen's d) of 0.185, between the "accuracy" of the participants and the "question type (the three types of mental models). The same calculation was also performed against the "accuracy" and the "task type" with a Cohen's d value of 0.0227, which might indicate that there did not exist such a correlation. The correlation might not appear, even we recruited a considerable

amount of people for our study. Therefore, we picked the larger Cohen's d as the measurement to calculate the required participants, as it guaranteed the effect at least. With this Cohen's d value, our single group design (only one group of subjects with only one treatment with two levels) required 71 participants with an alpha value smaller than 0.05 and a power score above 0.80, according to Robert and Randell's work [3]. This paper did not provide us with formulae to calculate the exact sample size requried for our study. Instead, it only offered us the sample sizes with the stepwisely listed effect size and number of repeated measures. The table with power at 0.80 and alpha at 0.05 only showed an effect size of 0.15 and 0.20, with no other value between them. Meanwhile, the number of repeated measures from this table only had values of 10 and 20, with no other value between them, also. Since our pilot study showed an effect size of 0.185, with 16 repeated measures, we obtained our sample size from the table with the largest effect size that was smaller than our value (0.15), and the largest number of repeated meausres that was smaller than our case (10). The crossed result from this table led us to a sample size of 71. This is a conservative estimate for the smallest number of participants required (the largest Cohen's d and number of stimuli from the table that are smaller than those in our study).

Another solution to this participant problem, is a rule of thumb already proposed by Brysbaert and Stevens in their work, which states that to satisfy a significant statistical power the number of stimuli multipled by the number of participants should be at least 1,600 [20]. This means that we need exactly 100 participants ideally, since our mental model questionnaire contains 16 questions.

Chapter 4

Results

We divided our participants into two randomly assigned groups: read-to-do and read-to-recall. This meant that there were two different levels of our studies, as each of the participants had to finish a questionnaire with 16 independent questions. A linear mixed-effect model is the best choice for our study, since our study is a within-subject repeated measure design, and traditional methods, such as ANOVA, are not able to fully cover the interaction terms [8]. In this chapter, we will discuss the participants first, and then describe the distribution of the data and models.

4.1 Participants

We published the same code and questionnaire for the main study. The experiment was hosted on Pavlovia, an online study hosting platform, which allowed people to upload experiment designed on PsychoPy and could automatically translate the PsychoPy-designed Python script into online compatible JavaScript script [26]. We found these participants via different social media platforms and direct references between participants. For the English version of the study, we posted our experiment advertisement (Appendix C) with the Pavlovia link to two reddit

channels: r/SampleSize and r/ProgrammingBuddies. Each of these two channels contributed approximately half of all the English participants. We also posted the same experiment advertisement to a free survey-hosting website SurveyCircle, but unfortunately 0 people participated from this website, as it had an internal ranking system which made newly posted surveys not able to appear at the first page. For the Chinese version of the study, we found that it was almost impossible to post our studies on one of their major online forums or social media, since registering an account on those platforms requires a Chinese phone number for personal identity verificiation. This imposed a limitation on our ability to reach to our participants. Rather, we used our relationship connections and found some IT specialists in some companies (human resource companies that use IT systems to digitize their paper documents) and asked them to help us refer our study to other people they know who understand concurrent programming. However, doing so may have introduced potential sampling biases, since these people are more likely to refer our study to their colleagues than the university students.

According to the report generated by Pavlovia, a total of 337 people visited our study, yet only 113 people finished it (86 from the English language group, and 23 from the Chinese language group, another 4 repetitive). Among these 113 finished sessions, only 85 generated the proper data worthy of analysis. We discarded 28 responses due to the following reasons: 1) Their answer on Question 4 (Number of concurrent programming courses finished) and Question 8 (Number of years of experience in concurrent programming) from the demographic questionnaire (Appendix D) showed little or no experience in concurrent programming. 2) They filled irrelevant comments during their code-reading task. We discarded the responses that matched either of these two criteria.

4.2 Data

4.2.1 Data Overview

The mean number of accurately answered mental model questions (accuracy) among all the participants was 9.51 (59.49%) of the 16 questions. We divided these results into different categories, according to the tasks assigned to the participants, the types of mental models the questions belonged to, and the languages the participants spoke. The average accuracies, classified by the task type (read-to-recall or read-to-do), showed that for those who were assigned to describe the purpose of the program (read-to-recall), the overall correctly answered questions was 9.46, which gives a rate of correctly answered questions of 59.10%, while the read-to-do group demonstrated a similar result, with a mean accuracy of 9.59 (59.92% correct). From the average accuracy between the two task groups we find a similar average accuracy, showing that the type of task assigned to each participant may not be a significant factor. However, this can be masked by the fact that our study has many interacting factors and these factors can play a significant effect on the overall accuracy as well. The same principle of interaction also applies to the average accuracies of all the following factors.

Of the 5 concurrent (execution model) questions, people answered an average of 2.78 correct (55.53% correct). We also found that among all the 5 program model questions people answered 3.32 correctly, which is 66.59% correct. Meanwhile, people answered the 6 situation model questions with 3.55 correct, which is 59.22%. The average of the program model question accuracy is much higher than the rest of the two categories. This may reflect the fact that the program model questions are easier to answer, and the program model itself requires less effort to be built. Since many studies have shown that any educated programmer is able to construct this model, we consider these questions to be removable in

any future studies [1][9][7]. Further analysis is required to determine whether the question type itself or an interaction between this and another factor caused this difference in accuracy.

The average accuracy by language showed a similar pattern in both groups. The English-speaking group showed a mean of 9.41 correctly-ansered out of 16 questions, with a 58.79% correctness ratio, while the Chinese-speaking group showed a mean of 9.85 correctly answered-question out of 16 questions, with a 61.61% correctness ratio. However, we cannot conclude that Chinese language plays a better role of the understanding as there may exist some potential sampling bias between language groups.

A simple analysis of the accuracy mean and standard deviation shows that the accuracies and standard deviations of the different questions vary widely. The mean accuracies of each question ranged from 0.27 to 0.98 (a value of 1 means everybody answered this question correct, while a value of 0 means everybody answered this question wrong). Meanwhile, the standard deviations also share a similar wide range, ranging from 0.152 to 0.503. The means and the standard deviations of each question answered by all the participants are shown in Table 4.1.

We found some visible yet not separable patterns, specifically that the participants answered the program model questions better than those categorized in the situation model and the execution model. Table 4.1 shows that the three most accurately answered questions were program model questions. However, participants answered two of the program model questions (Question 2 and 5) very incorrectly, as the mean accuracies of these two questions are of the lowest of all, which means almost nobody answered these questions correctly. We suspect that the "Money" in Question 2 (Does the program define the "Money" class?) is very similar to the notion of "Account" in Question 1 (Does the program define

question number	model type	mean	sd
1	Program model	0.98	0.15
2	Program model	0.32	0.47
3	Program model	0.85	0.36
4	Program model	0.84	0.37
5	Program model	0.35	0.48
6	Situation model	0.66	0.48
7	Situation model	0.71	0.46
8	Situation model	0.48	0.50
9	Situation model	0.41	0.50
10	Situation modell	0.58	0.50
11	Situation model	0.72	0.45
12	Execution model	0.72	0.45
13	Execution model	0.68	0.47
14	Execution model	0.49	0.50
15	Execution model	0.27	0.45
16	Execution model	0.47	0.50

Table 4.1: Mean and standard deviation of every each question, mean sorted

the "Account" class?); and that the object "amount" in Question 5 (Is the field "amount" represented by a floating point number?) was not fully specified to be the "amount" of the "Account" class, so some participants misunderstood the "amount" to be a general notion of money. Meanwhile, people answered the situation model and the execution model questions fairly equally well, as there does not exist any visible patterns between these groups. We will include the types of mental models as a factor into our interaction analysis, in order to find out if the mental model type alone contributed to this difference, or if the interaction between the mental model type and the type of task assigned to each participant played a more important role.

4.2.2 Box Plot Analysis on Distribution

We have drawn several box plots to analyze the distribution of the results. The box plot tells us the median, first and third quartiles, and extreme values using

the interquartile range (IQR) (Q3 + 1.5 * IQR or Q1 - 1.5 IQR) [12].

The accuracy distribution given by the type of task assigned to the participants is shown in Figure 4.1.

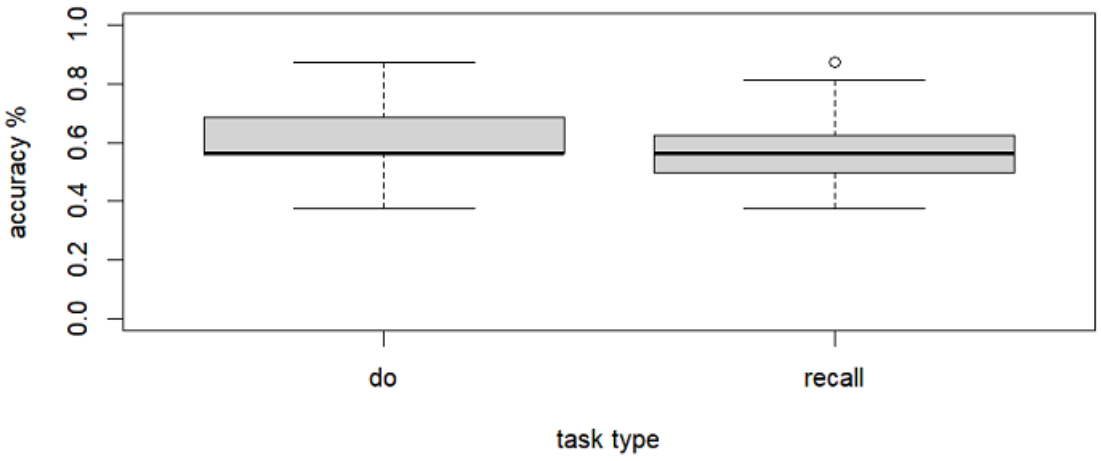

Figure 4.1: Accuracy distribution by task types.

Notice that the box plot in Figure 4.1 shows an outlier from the recall group. This outlier is not a reason for us to claim any data to be invalid, as it is possible and reasonable for anyone to answer all 16 questions correctly or wrongly. In addition, this box plot also shows the median accuracy between both groups are the same; however, we had a skewed accuracy distribution in the read-to-do task, while that of the read-to-recall task was more evenly distributed. Meanwhile, the range of accuracies between these two groups are almost identical, suggesting that participants presented with different tasks could perform equally well or badly answering the mental model questionnaire. This shows that the result from the read-to-do group is skewed to the low end, resulting a higher mean accuracy in this group (59.92% correct) compared to that in the read-to-recall group (59.10% correct). In the read-to-recall, group we found the median correctness is

also around 55%, but the range lies between approximately 50% and 65%. We cannot simply conclude that the read-to-do group answered the questions better than the read-to-recall group, because the accuracy between these groups has an overlapping pattern. As our study involves interactions among factors, we need to run an interaction analysis to tell the true difference.

Another box plot was drawn to represent the distribution between the accuracy and language the participants were speaking, as shown in Figure 4.2 below.

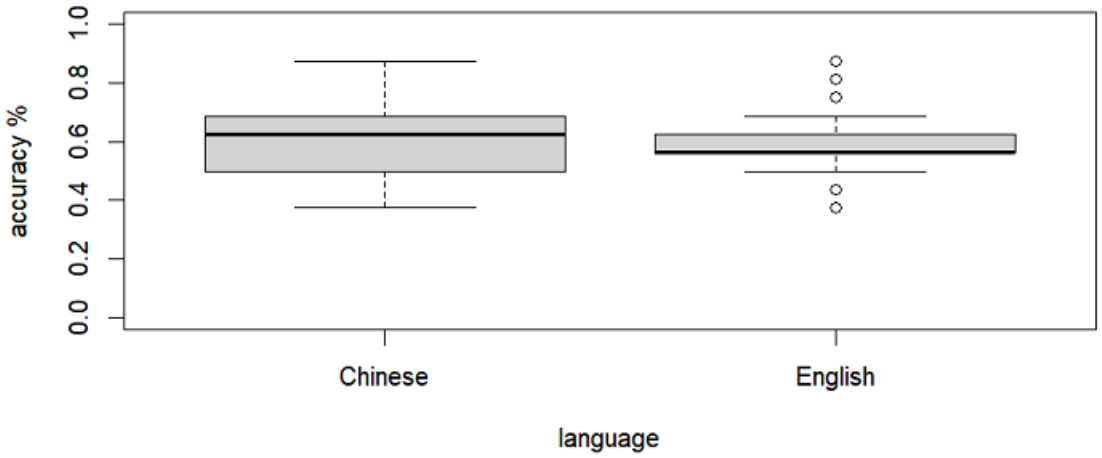

Figure 4.2: Accuracy distribution given by language

Figure 4.2 shows that the accuracy of the English speakers was distributed more tightly compared to that of the Chinese speakers, and that the Chinese speaker had a higher median accuracy. This difference in the distribution pattern might indicate that language possiblly played an important role in this difference. However, there are many different factors that might cause this difference. First, we have considerably more participants in the English-speaking group which may naturally lead to a tighter distribution. Second, there existed a sampling bias between these language groups so a significant difference of accuracy against this

factor might not accurately reflect the true effect of the language.

A box plot for the accuracy against the mental model type is also drawn. Figure 4.3 draws the box plot for all the three types of questions based on the mental model.

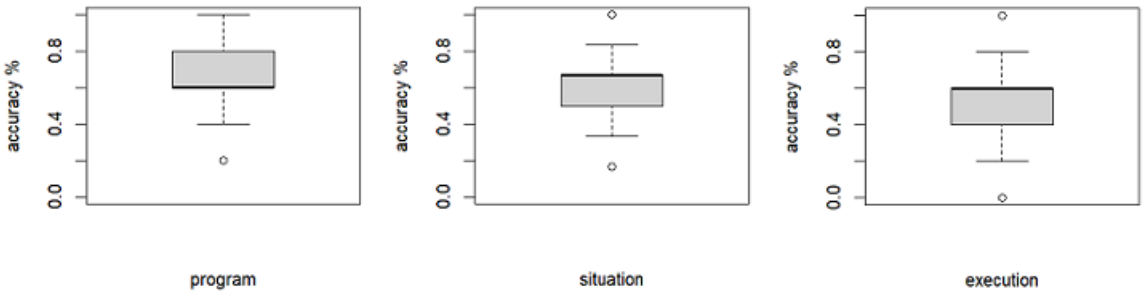

Figure 4.3: Accuracy distribution given by mental model types

From these graphs we observe that even though the program model questions seem to have been answered the best, yet the median of this category is lower than the situation model, and shares a similar value with the execution model questions. However, the median is accompanied by a skewed distribution to the low end, whch means many people answered these questions well. The distribution here seems to be very significant among the mental models, but further analysis is needed to understand if these differences are truly significant, because of the overlapping of the accuracy distribution among the three types of mental models.

4.3 Interaction Analysis

Since this is a crossed study between the task and question types, an interaction analysis is required to understand their relationship more fully. We need to use a linear mixed-effect model, as our mental model questionnaire presented questions

with interacting conditions. People may argue that using ANOVA can also be a viable approach to the categorical within-unit predictors, but ANOVA cannot handle cases where the same subjects were presented with the same stimuli [8]. In addition, using the linear regression model can help us build a logistic model, which fits better to our data as our accuracy is classified as Bernoulli trials (events with binary outcomes) [4].

We have configured the model to fit the accuracy data, categorized into three different types, according to the types of mental models (program model, situation model, and execution model). Since we did not give any information to the participants about the type of model of the questions when we presented the data to them, as this information is not coded into the experiment, we need to add the model type to each question in our data file. As we have previously mentioned, it is not necessary to scramble and randomize the order of the questions, according to Burhkardt's study [9]. Thus adding the model type was quite an easy task. We simply created an extra column in the datafile and inserted the mental model types into this column so that the question and mental model type are perfectly mapped.

Due to the fact that there exist some potential sampling biases, we decided to separate the analysis between the two language groups first. We selected the subsets of the data based on language to form two new data frames, one exclusive to English speakers, and another exclusive to Chinese speakers. After we have created these two data frames, we pulled out columns required for the model from each of these data frames and converted the categorical variables into *R*-style factors first, because the function *allEffects()* from the *effects* package required to build the effect table does not accept creating factors inside the model parameter [16]. However, to help demonstrate our statistical model, we choose to incorporate every possible calculation into the model code to demonstrate the details of the

models, and to maximize the possible interactions between the factors.

After we factorized the data and nominated the levels in the factors (read-to-recall vs. read-to-do in the task type, program model vs. situation model vs. execution model in the model type, and Chinese vs. English in the language), we used the *bglmer* method, a method included in the *blme* package which implements on the *lme4* package, to build our models. This package uses the Bayesian probabilities rule to estimate the linear model parameters with the conditional presence of another value or factor, making it more stable and helping us avoid singularities in logistic regressions [10]. We also used the *nlminbwrap* linear model control to offer alternative optimizers for tolerance of singularities [5]. These models could help us create a logistic regression line which is suitable for data prediction in different categorical groups.

For each language group, we built two models to analyze and compare for the best fit. The differences between these two models are that one of the models only has random intercepts, while the other model has both random intercepts and slopes. We compared the models to see if random effects from the factors (tasks and types of models) affected the random effect from the participants themselves [4]. From the random slope models, we sought the random effect of possibly both factors, including the task type, model type, and interaction between them (A*B = A + B + A:B in *R*). For the random intercept-only model, we included none of the factors to measure their random effects. On the other hand, for the random intercept and slope model, we wanted to see if the random effects of both factors (task types and model types), and the interaction term between these two factors, had any effect to the data. The general formula of the linear mixed-effect model with interaction is the following:

$$Y_i = \beta_0 + \beta_1 X_1 i + \beta_2 X_2 i + \beta_3 X_1 i X_2 i + e_i$$

Two β values, β_0 and β_2, are the y-intercepts for the baseline group and the offset for the alternative group, respectively. At the same time, β_1 and β_3 are the slopes for the baseline and the offset for the alternative, respectively. X_1i is the predictor, and X_2i is the dummy code, taking 0 or 1 to indicate the groups, 0 for baseline, and 1 for the alternative. In the random intercept-only model, we only allow β_0 and β_2 to be random, while in the random intercept and slope model we should allow all the β values to be random [4].

In our study, if we let the read-to-recall task to be the baseline, and the read-to-do task to be the alternative, then the formula can be converted into these two variations, depending on the value of the dummy code of X_2i, where the beta values are the intercepts and slopes for the mental model types (program, situation, and execution models):

$$Read\_to\_recall : Y = \beta_0 + \beta_1 X_1 i + e_i$$

$$Read\_to\_do : Y = (\beta_0 + \beta_2) + (\beta_1 + \beta_3) X_1 i + e_i$$

This formula is the most basic form of a linear mixed-effect model with a 2x2 experiment design [4]. In our study more terms were added to our formula when we built the R model, since our study is a 3x2 design (3 mental models, 2 task types), so that we can generate more linear formulas based on the X values. Furthermore, our linear formula may not be strictly limited to the form of one-dimensional linear equation, since our data is binomial and classified into different categories. Instead, our data can fit better into a logistic model, which is also a linear model. Doing so may further alter the expression of our linear formula, but the fundamentals remain the same as we need the dummy code X_2i to make the distinctions determined by the tasks assigned to the groups.

After we finished these two models, we ran ANOVA tests on every single model to obtain the significant factors, then we ran an ANOVA comparison to see if these

two models (random effects from the factors) are significantly different from one another [19] [14]. If there does not exist any notable difference (p = 1) we will only show the results from the random intercept-only model, as the random effects of the slope do not contribute anything more to the model fit.

4.3.1 English-Speaking Group

We started with the English-speaking group with the following models (random intercept only model first, then random intercept and slope model):

$$
\begin{aligned}
&bglmer(as.factor(accuracy) \sim as.factor(task\_type) * as.factor(model\_type) + \\
&(1|as.factor(subject\_ID)), \\
&family = binomial, \\
&control = glmerControl(optimizer = "nlminbwrap"), \\
&data = subset(data, Language == "English"))
\end{aligned}
$$

$$
\begin{aligned}
&bglmer(as.factor(accuracy) \sim as.factor(task\_type) * as.factor(model\_type) + \\
&(1 + as.factor(task\_type) * as.factor(model\_type)|as.factor(subject\_ID)), \\
&family = binomial, \\
&control = glmerControl(optimizer = "nlminbwrap"), \\
&data = subset(data, Language == "English"))
\end{aligned}
$$

After we successfully let R construct these models, we first examined the summaries of the models. This is achieved by the "summary()" method. From both the summaries of these models we observed from the fixed effects that both the intercepts (the baseline with the read-to-do task and program model) and the execution (read-to-do task and execution model) preserve significant effects (p <0.05), while all the rest of the terms do not (p >0.05). All the value terms in both the linear models above show that there exists either a strong or sizeable

correlation between any of the terms or the intercepts, with the absolute values ranging from 0.357 to 0.730 in both models. However, the correlation data are not sufficient to understand the true relationships because they are all based on the categorical variables; therefore, we need some other analysis to understand our model. Here we choose the ANOVA analysis on the Chi-Square to test the type-II and type-III analysis-of-variance tables and use the ANOVA comparison of the Chi-Square to find the differences between the models (R can decide if the Chi-Square or F-test, depending on the data and model types fed into the ANOVA comparison method) [19] [14].

The ANOVA analysis of both of the models, which was used to analyze the significant factor, showed that the mental model type was a significant factor ($p < 0.05$) while the task factor and the interaction between the task factor and mental model type were not the key factor ($p > 0.05$) The ANOVA comparison between these two models yielded a p-value of 1, which indicates that both of the models fit the data equally well, and the random effects on the slopes had no effects on the outcome. Therefore, we will only show the outcome from the random intercept only model.

To understand how exactly the significant factor brings the effect to the outcome, we adapted the analysis from the Chapter 4 of Barr's work *Learning statistical models through simulation in R: An interactive textbook*, using the effect package from R to build the effect table, and plotted the effects into a graph to visualize the data [4]. The effect table with the marginal mean calculated, and the effect graph is shown below:

Task / Model	Program	Situation	Execution	Marginal mean
Read-to-do	0.594	0.619	0.471	0.561
Read-to-recall	0.588	0.580	0.533	0.567
Marginal mean	0.591	0.600	0.502	

Table 4.2: Effect table for English speakers' group, marginal means listed aside

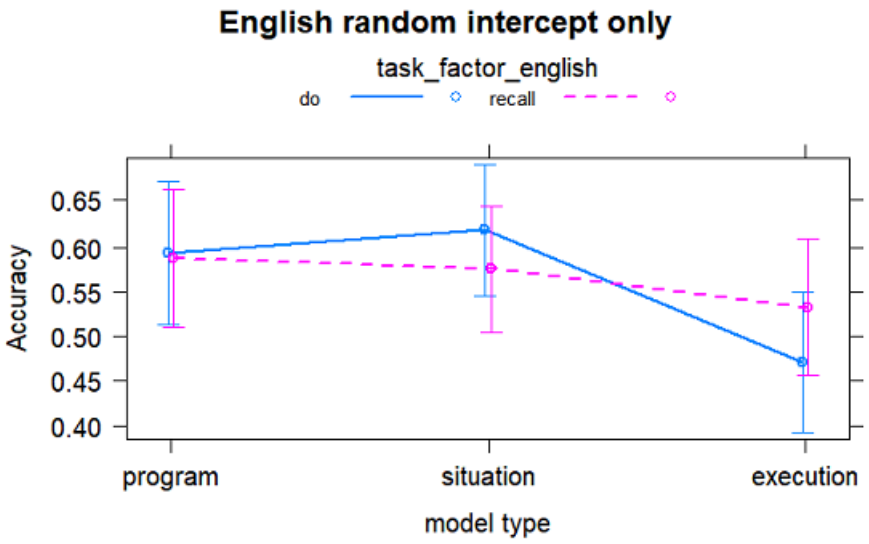

Figure 4.4: Effect graph for English speakers' group

From the table and the plot above, we can see there does not exist any significant difference between the two types of tasks (marked by the dotted and solid lines) as the data within the tasks and the mental model type are highly overlapping. Also, within each type of mental model, the distributions of the accuracy mean between two types of tasks are highly overlapping, as we can find in the bars (the range of accuracy of each question within this mental model type) stretching from the mean points. However, we observed that the marginal means of the execution questions is significantly lower than the rest of the two categories. Meanwhile, in the English-speaking group, there does not seem to exist any major difference between the program model questions and situation model questions. We conclude that in the English-speaking group the type of task does not have a

strong effect on the overall accuracy, but the question model type, especially the execution model, has a significant impact on the overall accuracy.

4.3.2 Chinese-Speaking Group

We did the same analysis on the Chinese-speaking group as we did in the English-speaking group using the same model construction techniques mentioned in the English-speaking group. The summaries of the random intercept-only model show that the intercept (baseline of read-to-do group and program model questions), situation model (read-to-do task), and execution model (read-to-do task) have key differences from the baseline, while the random intercept and slope model shows only the intercept, and the execution model shows a significant difference ($p < 0.05$). The ANOVA comparison of these two models yields a p-value of 1, which means both of the models fit the data equally well. Therefore, we will only show the outcome from the random intercept only model.

The ANOVA analysis of the random intercept model shows that the question model type plays a significant role ($p < 0.05$) while the other two factors (type of task and the interaction between the type of task and the type of model) have a non-significant impact to the overall accuracy ($p > 0.05$). The effect table and the effect plot are shown below:

Task / model	Program	Situation	Execution	Marginal mean
Read-to-do	0.869	0.632	0.511	0.671
Read-to-recall	0.919	0.542	0.699	0.710
Marginal mean	0.594	0.587	0.590	

Table 4.3: Effect table for Chinese speakers' group, marginal means listed aside

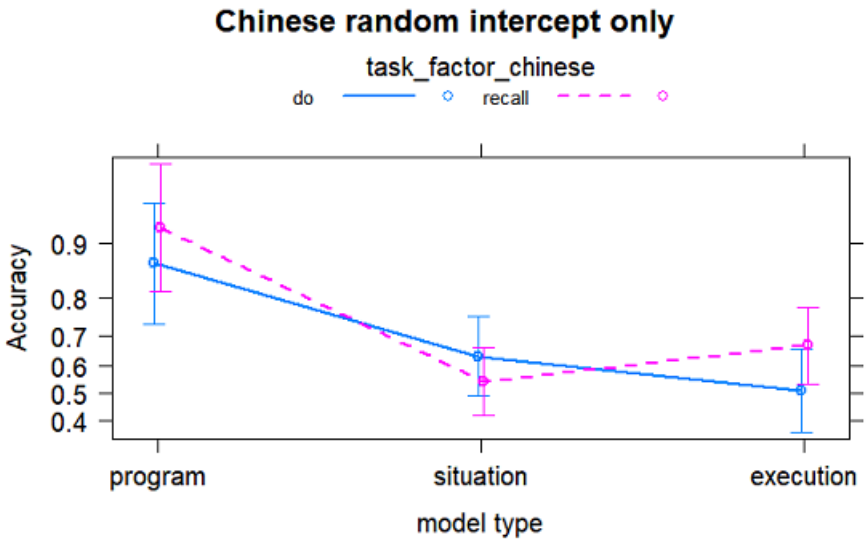

Figure 4.5: Effect graph for Chinese speakers' group

From the table and plot above, we found that within the Chinese-speaking group, the type of task does not seem to have significant differences, and the mean variations within each mental model category show overlapping indicated by the mean distribution line stretching from the mean points. However, the people answered the questions for the program model much better than the rest of the two categories. At the same time, people in this language group answered the situation and execution model question fairly equally well, with very close marginal means from the effect table. It does not show the same pattern as the English-speaking group, and we cannot draw a conclusion simply from the difference between these groups because there exists a potential sampling bias in this group. However, at least we are confident that the type of task has no significant impact on the accuracy within this group, and we still need some further studies to understand the differences between the two language groups, especially the uneven distribution between the program model questions.

4.3.3 Both Groups Blended Together

Even though we have a visible difference between the language groups with their answers to the program model questions, we still want to see the data that represents the population instead of limiting ourselves to the language group. Therefore, we built the third set of models, with data from both language groups mixed together using the same construction blocks.

The summaries of these two models show that both the intercept and the execution model impose a significant impact on the accuracy, but we need to look into the effect to find which factor is the most significant, since the execution model is only a subset of the mental model category.

The ANOVA comparison between these two models yielded a p-value of 1, meaning that both of the models fit the data equally well. Therefore, we only focused on the random intercept model.

The ANOVA testing on the random intercept model showed that only the mental model brings a significant impact on the accuracy ($p < 0.05$) while the task and the interaction between the task and mental model type are not significant ($p > 0.05$). Again, we need the effect table (with marginal means) and the effect plot to help us visualize and understand the data:

Task / model	Program	Situation	Execution	Marginal mean
Read-to-do	0.655	0.621	0.480	0.585
Read-to-recall	0.676	0.570	0.569	0.605
Marginal mean	0.666	0.600	0.530	

Table 4.4: Effect table for both language groups, marginal means listed aside

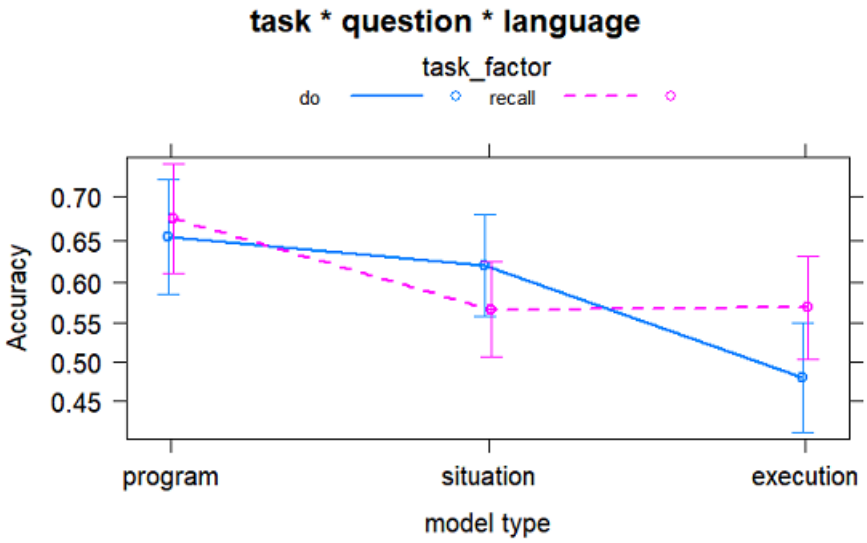

Figure 4.6: Effect graph for both language groups

We found that there does not seem to exist a very significant difference between the read-to-do and read-to-recall groups, but we see a gradual decline in the accuracy, and this decline shared the similar pattern with our box plots above. People answered the program model the most accurately, followed by the situation model in the middle, and the execution model the worst, from the marginal means generated from the effect table. We are confident that there does not seem to exist any sign of a significant impact of tasks on the accuracy, whereas the question model type is very significant, due to the lines between the two task groups are crossed. On the other hand, the accuracy classified by the mental model type within each task group is significantly different among each other.

4.3.4 Correlations Between Correct Answers and Personal Reviews

We found the overall average self-assessment score is 58.8% and personal estimated peer assessment score is 60.3%. To find if the assessment scores reflect the accuracy

with which participants answered the questions, we need to run correlation tests on the result data. We built correlation testings using the Spearman method, which correlates normally distributed data in a parametric way [24]. The correlation tests take the following format in R:

$$cor.test(data[data\$model\_type == "program",]\$self,$$
$$data[data\$model\_type == "program",]\$accuracy,$$
$$method = "spearman", exact = FALSE)$$

Using this method, we found that in almost of the cases there seems to exist very small or no correlation, using Cohen's criteria from his publication in 1988 [11]. In Cohen's publication, he categorized the correlations into three different categories: r = 0.1 is weak, r = 0.3 is moderate, and r = 0.5 or above is strong [11].

The correlation of between the self-assessment or personal estimated peer assessment and the accuracy can be found in the table below.

	self	peer
all questions	0.009	0.081

Table 4.5: Correlation table of all questions, task, question, and language type not considered

Table 4.5 shows some tiny, almost negligible correlations between the assessment and the accuracy. However, we needed to look deeper into this data to decompose it, based on task type, question type, and language type, respectively.

When analyzing the self-assessment by tasks, we found people from the read-to-do task rated themselves higher on both the self and personal-estimated peer assessment. Both of the ratings from the read-to-do group are higher than 60% (self: 60.85%, peer 64.78%), whereas the rating from the read-to-recall group shows a similar value below 60% (self: 56.98%, peer: 56.38%).

From Table 4.6, we noticed that in the read-to-do group there exists a weak association with the self assessment. Yet in the read-to-recall group the association is negligible.

All questions included	self	peer
recall	-0.090	0.043
do	0.130	0.118

Table 4.6: Correlation table: task type vs. accuracy

We also divided the serial questions into program model questions and situation model questions, and noticed that there exists a small correlation between the self assessment and accuracy in the situation model. Table 4.7 shows the calculated data.

	self	peer
program	0.023	0.031
situation	-0.145	0.052
execution	0.060	-0.062

Table 4.7: Correlation table: mental model type v.s. accuracy

We noticed that there exists a negative yet weak correlation between the situation model question accuracy and the self assessment. This is expected because normally people should rate themselves higher if they have a higher accuracy in this type of questions, demonstrating a level of confidence. This confidence analysis can be included in the future studies.

When we classify the participants according to the language, the assessment score shows a different pattern. The English-speaking group has a higher personal estimated peer assessment score, yet the Chinese-speaking group has a higher self-assessment score. We also did another correlation analysis between the language and the accuracy. Table 4.8 still shows a generally weak or negligible association

betwen the language and the accuracy, but we still find that self assessment of the Chinese-speaking group shows a relatively stronger yet still weak association.

All questions included	self	peer
English	-0.014	0.096
Chinese	0.025	0.013

Table 4.8: Correlation table: language v.s. accuracy

Nevertheless, our correlation analysis result shows that there does not exist any strong or moderate correlation between the task assignments, type of questions, and language group. This result is different from Bidlake's pilot study as it shows that there exists a strong or moderate correlation between the accuracy and self-estimation score. However, in Bidlake's article she claimed that her study is only a pilot study and only recruited eight participants, which lacks a sufficient statistical power [6]. It is also possible that the differences are due to the fact that Bidlake's work asks participants to determine if there exists any race conditions, whereas in our study we asked participants to answer comprehension questions.

Chapter 5

Conclusions and Reflections

5.1 Conclusion

From our data analysis, we found that the task type does not bring a significant impact to the accuracy. On the other hand, the type of question is very significant. People answered the program model questions much better than the situation and execution model questions, because any well-educated programmer is able to build the program model [1]. However, a lower situation and execution model question accuracy may indicate that there are some novice participants who can only build the program model, but not the situation and execution model, and has already been shown in Burkhardt's work [9]. Not all programmers have a good understanding of concurrent programming, and we expect that concurrent programming is difficult to understand due to its nature, since it introduces another extra level (execution model) to the program [1].

We also found that the correlations between the personal assessments and the accuracy is rather small, which means that the participants failed to rate their concurrent programming skills properly. As this rating is subjective, this data has a tendency of being inaccurate, and we need to design some other measurements,

such as another questionnaire to determine the correlation between the accuracy and the expertise level in concurrent programming.

We also attempted to analyze the responses such as comments and expertise level based on Question 4 (Number of concurrent programming course finished) and Question 8 (Number of years of experience in concurrent programming) from the demographic questionnaire, but we found that it was very difficult to draw conclusions based on this data, because there still does not exist any well-established method to analyze the expertise level based on comments in mental model studies [7]. We also found that the demographic questionnaire only partially reflected the true expertise of a programmer, because we observed that many of the IT specialists (especially in the Chinese language group) reported a similar expertise level in concurrent programming, and many of them reported having little experience, yet this difference is not determinate as there exists a cultural factor. On the other hand, the PsychoPy version we used in the study does not offer us a suitable method of recording this type of data, as the slider module in this release only allows categorical inputs. At the same time, the only other module that allows a rating input is the textbox where people can enter anything possible. We were not able to further analyze this part (it is also not a major focus of our study).

Since we found that there is no significant difference between the "read-to-recall" and "read-to-do" tasks in the construction of the mental model in the understanding of the concurrent program, we speculate that there may exist some overlap between the "read-to-recall" and the "read-to-do" task. The participants have to read the code and understand what the code is doing in both cases first, so they can write the purpose or identify the bugs. This is especially true in the "read-to-do" task since people need to understand the purpose of the code first and then to find bugs in the program in order to fix the program according its purpose. This overlap can be one of the possible reasons that the type of the task

is not a significant factor. It is also possible that the program was too simple, in order to limit the time each participant spends on our experiment. It would be possible to use a more complex program so that we can observe a difference, but doing so means we need to pay the participants more to compensate them for their time.

Even though the ANOVA analysis shows a very significant effect of language against the accuracy when we included all the questions, it cannot guarantee a solid conclusion that language difference is another key contributing factor in concurrent programming comprehension. Since there exist potential biases in the sampling process as previously mentioned, we cannot conclude whether language is or is not a significant factor. Further studies are required to resolve this issue. Yet the result shows that, within each language group, we find that the task assigned to each participant is not the key factor.

Nevertheless, these analyses still demonstrate that regardless of language, the type of task does not make a very significant impact on how people understand all the three types of models (program model, situation model, and execution model). Contrary to what we have previously assumed, our experiment result showed that only the type of questions have significant effect on the number of accurately answered questions. Neither the type of task nor the interaction between the type of task or type of mental model had an impact. Therefore, we partially reject our original hypothesis, as we found that there is a significant difference among differernt mental model types; yet the difference in tasks shows little impact to the mental model construction when people approach concurrent programming.

5.2 Reflection

As our analysis have shown, we partially rejected our null hypothesis and stated that the type of task has no effect on the mental model construction during the concurrent programming. There are also weak or no correlations between the accuracy and different kinds of assessments, regardless of whether they are self-assessments or personal estimated peer assessments. We admit that this study is not perfect because a similar study has never done before, which means we have limited comparison and learning from other studies [7]. This study concluded that debugging may not result in a better approach to mental model construction when programmers approach concurrent programming. This study also showed that the mental model construction of concurrent programming in terms of question accuracy, has tiny or negligible correlations with the assessments of expertise including both self and personal estimated peer assessments. It has established the first evidence of the mental model construction (execution model) analysis in concurrent programming. However, we found several issues and suggesst possible improvements for future studies.

First, we admit that finding particpants with explicit knowledge of concurrent programming is not easy, since concurrent programming are very often used as a mean of program optimization and performance boost [2]. This means that programmers will focus more on the fields they specialized in but not concurrent programming alone. As a result, far fewer people actually possess concurrent programming knowledge since most of their developing tools have already been optimized with concurrency released within the libraries, such as *Apache Hadoop*. In addition, concurrency is not always a requried knowledge in programmers' education, and the machine-level execution is something that programmers tend to forget and ignore. As a result, it is relatively more difficult to find enough participants to maintain a solid statistical power.

Second, we found participants are highly likely to quit and not finish the experiment if the code is complicated, such as the code we used in the first version of our pilot study. Complicated programs are more difficult to remember and participants may feel frustrated even if we present them as few questions as possible. The number of questions would grow at least linearly with the complexity of the program. In fact, the feedback from several participants showed that they prefer to answer questions with the presence of the code on the side. A scenario like this is actually closer to the real-life programming, as programmer can simply go through different code snippets but not to remember all the features. However, since our study is a mental model study, we need to ask these questions without presenting the code because we want to analyze the mental model built in participants' minds, presenting the code alongside the questions can interfere such a process.

Third, we also noticed that there might exist a sampling bias between and within the language groups, due to different policies among the programming forums and communities. Many popular programming communities, such as StackOverflow (Stack* family of forums), never allow people to post links to their studies because they cannot guarantee that these studies are real studies or simply advertisements. In addition, many of the Chinese discussion forums require a Chinese phone number to register an account to create posts. This greatly limited our access to the programming communities, as the Chinese phone number is region-exclusive and not available to us. As a result, we recruited our participants from less professional communities on Reddit. Thanks to its huge user population we managed to find enough participants to fulfill the statistical power. Ideally, we need to include people from other language and ethnic groups to perform analysis with language. However, this is likely to make a study overcomplicated. A better approach is to design a study without the interference of language at all,

such as program modification as the only "language" people need to know is the programming language itself.

Nevertheless, our study has demonstrated that the task assigned to the participants is not important when they try to build the mental model to understand the concurrent programming, and we hope that future studies can find the key factors that we cannot find.

5.3 Future study

We admit that our study is not perfect, and the following aspects can be improved:

1) Our study only asks people to read our program instead of letting them try to directly modify and run the program, which is more common in the software developing cycle. This is highly limited by the fact we published our study online and the online platforms have their own access restrictions to the code compilers. On the other hand, even though there are many online code compilers online, but their compiler configurations are different, making it difficult to choose the optimal one. As a result, we may need to recruit participants locally, set up an experiment environment, and let people debug and run the code.

2) We could in a future study recruit participants from different sources with minimum sampling bias. This can be difficult since people use different social media in different countries. It would be better to find similar forums with almost the same type of population. However, doing so requires an understanding of different social media and even the different languages used in that group. Also, we need to find other methods to award the participants in different countries because this may also be related to the banking system the target country has, as some countries may require some information exclusive to their territories and prevent us from finding and giving awards to the participants from those regions.

3) We need to carefully design our code sample so that we can give the participants different and non-overlapping tasks, while all the groups use the same program. Burkhardt's study has already demonstrated that it is viable that there to have little overlap between the tasks, even though all the participants read the same program [9]. One alternative approache woule be to let the "read-to-recall" group describe the purpose of the program, while let the "read-to-do" group make a list, or draw a diagram that shows the step-by-step machine execution patterns over certain lines from the same code. We recommend the concurrent idioms summarized by Lin and Dig in their work *CHECK-THEN-ACT Misuse of Java Concurrent Collections*, as their idioms are very versatile but difficult enough that they are also helpful to determine participants' expertise levels [18].

4) Since we noticed that there are little or negligible correlations between the personal assessments and the actual number of correct answers in the mental model questionnaire, we would like to add the analysis of these personal assessments into future study. A new study could use a questionnaire to ask participants questions instead of making them leave their own ratings. This analysis of assessment confidence can be developed into a standalone study and understand how well related the personal assessment and the concurrent programming are. It can also provide concurrent programming learners and college students a better guide to their realization of their skills.

5) As we noticed that people answered the program model questions much better, we find it possible to improve our mental model questionnaire in the following two ways. We can either remove those questions in the program model part that are too easy. It is also possible to add difficulty weights to the questions, but Burkhardt mentioned in his paper that it is difficult to calibrate the difficulty of the questions as calibrating difficulty can make questions less comparable to each other especially in complex programs [9]. Extensive works and research must

be done before designing the questionnaire so that the difficulties of questions in different mental models are well calibrated, and this can be reflected in the pilot study phase if we can rate the accuracies.

6) Even though there is no standard for us to fully read, understand, and assess the comments and feedbacks from the participants, it still possible that we could collect this information, and analyze participants' comments when reading the program. Perhaps we can identify some problems, or other key factors that contribute to the formation of the execution model.

References

[1] Eric Aubanel, *Parallel Program Comprehension*, PPIG, 2020, pp. 8–16.

[2] Eric Aubanel, *Elements of parallel computing*, CRC Press, Talor & Francis Group, ISBN: 978-1-4987-2789-1, 2017, p.1 pp. 19–20, 24–26.

[3] Barcikowski, Robert S.; Robey, Randall R., *Sample Size Selection In Single Group Repeated Measures Analysis.*, 69th Annual Meeing of American Educational Research Association, Chicago, IL, Apr. 1984

[4] Barr, Dale J., *Learning statistical models through simulation in R: An interactive textbook*, Version 1.0.0, Chapter 4, 5, 6, 7, 8, 2021, Retrived from: https://psyteachr.github.io/stat-models-v1.

[5] Douglas Bates, Martin Mächler, Ben Bolker, Steve Walker, *Fitting Linear Mixed-Effects Models Using lme4* Journal of Statistical Software, vol. 67, No. 1, pp. 1-48, 2015, DOI: 10.18637/jss.v067.i01

[6] Leah Bidlake, Eric Aubanel, Daniel Voyer *Pilot Study: Validation of Stimuli for Studying Mental Representations Formed by Parallel Programmers During Parallel Program Comprehension*, PPIG, 2022.

[7] Leah Bidlake, Eric Aubanel, Daniel Voyer, *Systematic literature review of empirical studies on mental representations of programs*, Journal of System and Software, Volume 165, ISSN 0164-1212, 2020, pp. 1–15.

[8] Markus Brauer, John J. Curtin, *Liner Mixed Effect Models and the analysis of Nonindependent Data: A Unified Framework to Analyze Categorical and Continuous Independent Variables that Vary Within-Subjects and/or Within-Items*, American Psychological Association, included in Psychological Methods, Volume 23, No. 3, 2018, pp. 389-411, DOI: http://dx.doi.org/10.1037/met0000159.

[9] Jean-Marie Burkhardt, Françoise Détienne, Susan Wiedenbeck, *Object-Oriented Program Comprehension: Effect of Expertise, Task and Phase*, Empirical Software Engineering, 7, Kluwer Academic Publisher, 2002, pp. 115–156.

[10] Yeojin Chung, Sophia Rabe-Hesketh, Vincent Dorie, Andrew Gelman. Jingchen Liu, *A nondegenerate penalized likelihood estimator for variance parameters in multilevel models*, Psychometrika, Springer, vol. 78, No. 4, pp. 685-709, 2013, DOI: 10.1007/s11336-013-9328-2

[11] Jacob Cohen, *Statistical power analysis for the behavioral sciences*, 2nd edition, Hillsdale, N.J., L. Erlbaum Associates, 1988

[12] Williamson, D., Parker, RA., Kendrick, Juliette. *The box plot: A simple visual method to interpret data.* Annals of internal medicine. 110. 916-21. 1989. 10.1059/0003-4819-110-11-916.

[13] Peter Dixon, *Models of accuracy in repeated measure design*, Journal of Memory and Language, ScienceDirect, Elsevier Inc., 2008, pp. 447–456.

[14] Andy Field, Jeremy Miles, Zoë Field, *Discovering Statistics Using R*, SAGE Publications Ltd, p.245, 2012.

[15] John Fox, *Effect Displays in R for Generalised Linear Models*, Journal of Statistical Software, 8(15), pp. 1-27, 2003, DOI: 10.18637/jss.v008.i15

[16] John Fox, Sanford Weisberg, *An R Companion to Applied Regression, 3rd Edition*, Thousand Oaks, CA, 2019, https://socialsciences.mcmaster.ca/jfox/Books/Companion/index.html

[17] Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea, *Java Concurrency in Practice*, Pearson Education, Inc, ISBN: 0-321-34960-1, 2006, printed at Courier Stoughton in Stoughton, Massachusetts, 2010, pp 5-6.

[18] Yu Lin, Danny Dig *CHECK-THEN-ACT Misuse of Java Concurrent Collections*, IEEE Sixth International Conference on Software Testing, Verification and Validation, ISBN: 978-0-7695-4968-2, Luxembourg, 2013.

[19] Thomas Lumley, *Analysis of complex survey samples*, Journal of statistical software, vol.9 pp. 1-19, 2004.

[20] Brysbaert M., Stevens M, *Power Analysis and Effect Size in Mixed Effects Models: A Tutorial*, Journal of Cognition, 1(1): 9, 2015, pp. 1–20.

[21] Nancy Pennington *Stimulus Structures and Mental Representations in Expert Comprehension of computer Programs*, Cognitive Psychology 19, 1987, pp. 295–341.

[22] Venkatesh Prasad Ranganath, John Hatcliff, *Slicing Concurrent Java Programs using Indus and Kaveri*, International Journal on Software Tools for Technology Transfer 9(5):489-504, DOI: 10.1007/s10009-007-0043-0, October 2007.

[23] Caitlin Sadowski, Andrew Shewmaker, *The Last Mile: Parallel Programming and Usability*, FoSER 2010, November 7–8, Santa Fe, New Mexico, USA, ACM: 978-1-4503-0427-6/10/11, 2010, pp. 1–5.

[24] David Sarmento, *A language, not a letter: Learning Statistics in R*, Chapter 22: Correlation Types and When to Use Them, https://ademos.people.uic.edu/Chapter22.html

[25] Roger C. Schank, Robert P. Abelson, *Script, Plans, Goals and Understanding*, Lawrence Erlbaum Associate, Hillsdale, New Jersey, Distributed by the Halsted Press Division of John Wiley and Sons, New York, Toronto, London, Sydney, ISBN: 0-470-99033-3, 1977, pp. 36–138.

[26] Peirce, J. W., Gray, J. R., Simpson, S., MacAskill, M. R., Höchenberger, R., Sogo, H., Kastman, E., Lindeløv, J., *PsychoPy2: experiments in behavior made easy*, Behavior Research Methods, ISBN: 10.3758/s13428-018-01193-y, 2019.

Appendix A

Code

```
class Account {
private int amount;

public synchronized int takeMoney(int a) {
    while ((amount - a) < 0) {
        try {
            wait();
        } catch (InterruptedException e) {  }
    }
    amount = amount - a;
    return amount;
}

public synchronized int addMoney(int a) {
    amount = amount + a;
    notifyAll();
    return amount;
```

```
    }

    @Override
    public synchronized String toString() {
        return Integer.toString(amount);
    }
}

class Employer implements Runnable {
    private Account save;

    public Employer(Account account) {
        save = account;
    }

    public void run() {
        save.addMoney(90);
    }
}

class Employee implements Runnable {
    private Account save;

    public Employee(Account account) {
        save = account;
    }
```

```
    public void run() {
        save.takeMoney(10);
        (new Account()).addMoney(10);
    }
}

class Business {
    public static void main(String[] s) {
        Account savings = new Account();
        Runnable Employer = new Employer(savings);
        Runnable Employee = new Employee(savings);
        new Thread(Employer).start();
        new Thread(Employee).start();
        new Thread(Employee).start();
    }
}
```

Appendix B

Mental Model Questionnaire

The correct answer to each question is highlighted.

Part 1: Program Model Questions

1. Does the program define the "Account" class? (Y/N) **Y**
2. Does the program define the "Money" class? (Y/N) **N**
3. Does the program define the "Employer" class? (Y/N) **Y**
4. Does the program define the "Employee" class? (Y/N) **Y**
5. Is the field "amount" represented by a floating point number? (Y/N) **N**

Part 2: Situation Model Questions

1. Does Account::takeMoney() check if the field "amount" is below 0? (Y/N) **Y**
2. Will Account::addMoney() notify someone if it has successfully received the amount? (Y/N) **Y**
3. Is the method "takeMoney()" part of the "Employee" class? (Y/N) **N**

4. Is the field “amount” shared to either “Employer” or “Employee”? (Y/N) **N**

5. Will the field “amount” always stay above 0? (Y/N) **Y**

6. Does the Employee add his/her money taken from the Employer’s account into another account not accessible to the Employer? (Y/N) **Y**

Part 3: Execution Model Questions

1. Are both methods of the “Account” class thread safe? (Y/N) **Y**

2. When two threads launched with the “Employee” taking away the money at the same time, will both of the actions be registered, resulting a total withdraw value of 20? (Y/N) **Y**

3. Does the Employee have to wait for the Employer to add money at any moment? (Y/N) **N**

4. Can the Employer add money into the account if the Employee has not finished taking away money? (Y/N) **N**

5. Is “80” the amount remaining after all the threads have finished their tasks? (Y/N) **N**

Appendix C

Experiment Advertisement

Public Announcement

GET A GIFT CARD FOR CONTRIBUTING TO JAVA CONCURRENCY RESEARCH

People needed for a study of concurrent programming

Please help us complete a mental model study concerning the understanding of java concurrent programming. Many people feel that concurrent programming is hard, and we want to understand why it is so difficult. The study is designed by a graduate student in the Faculty of Computer Science at the University of New Brunswick.

The study will last half an hour or less. By contributing to our study, you can get an e-gift card if you finish this experiment in one sitting. There are no other obligations for this study, and you can choose to start your study any time at your convenience.

People with experience in Java (especially Java Threads) are needed. We want:

College Student	College Professors
Programmers	Other people with related background

1. Please use a newer version of web browser, older browsers may not support the code module.

2. All participants must be over 19 years old.

3. If you meet these qualifications, click on this link to participate in our study pavlovia link. The experiment is required to be finished in one continuous sitting.

4. E-gift cards will be given to you later in August. We will not disclose any information at any time.

Appendix D

Demographic Questionnaire

1. Age (in number of years):
2. Are you currently a student? (Yes or No):
3. Number of programming course finished:
4. Number of concurrent programming course finished:
5. Highest level of education achieved:
6. Number of years since you first come to know Java:
7. Number of years of programming experience in Java:
8. Number of years of experience in concurrent programming:
9. Rate your level of expertise in Java concurrency, between 0 and 100 (0 = total lack of knowledge and 100 = total expert):
10. Rate your level of expertise in Java concurrency among peers, between 0 and 100 (0 = total lack of knowledge and 100 = total expert):
11. Email address for the gift card (optional):

Appendix E

Statistical Analysis R Scripts

1.1. **Random intercept only** model for **English-speaking** subjects

```
bglmer(as.factor(accuracy) ~
    as.factor(task_type) * as.factor(question_type) +
    (1 | as.factor(subject_id)),
    family = binomial,
    control = glmerControl(optimizer = "nlminbwrap"),
    data = data_english)
```

1.2. **Random intercept and slope** model for **English-speaking** subjects

```
bglmer(as.factor(accuracy) ~
    as.factor(task_type) * as.factor(question_type) +
    (1 + as.factor(task_type) + as.factor(question_type)
    | as.factor(subject_id)),
    family = binomial,
    control = glmerControl(optimizer = "nlminbwrap"),
    data = data_english)
```

2.1. **Random intercept only** model for **Chinese-speaking** subjects

```
bglmer(as.factor(accuracy) ~
    as.factor(task_type) * as.factor(question_type) +
    (1 + | as.factor(subject_id)),
    family = binomial,
    control = glmerControl(optimizer = "nlminbwrap"),
    data = data_chinese)
```

2.2. **Random intercept and slope** model for **Chinese-speaking** subjects

```
bglmer(as.factor(accuracy) ~
    as.factor(task_type) * as.factor(question_type) +
    (1 + as.factor(task_type) + as.factor(question_type)
    | as.factor(subject_id)),
    family = binomial,
    control = glmerControl(optimizer = "nlminbwrap"),
    data = data_chinese)
```

3.1. **Random intercept only** model for **all** subjects, language included as a factor

```
bglmer(as.factor(accuracy) ~
    as.factor(task_type) * as.factor(question_type) *
    as.factor(language)+
    (1 + | as.factor(subject_id)),
    family = binomial,
    control = glmerControl(optimizer = "nlminbwrap"),
    data = data_chinese)
```

3.2. **Random intercept and slope** model for **all** subjects, language included as a factor

```
bglmer(as.factor(accuracy) ~
    as.factor(task_type) * as.factor(question_type) *
    as.factor(language)+
    (1 + as.factor(task_type) + as.factor(question_type) +
    as.factor(language) | as.factor(subject_id)),
    family = binomial,
    control = glmerControl(optimizer = "nlminbwrap"),
    data = data_chinese)
```

Appendix F

Previous Code

```
class Book{

    private final String title;
    private final String id;

    Book(String Title, String ID){
        title = Title;
        id = ID;
    }

    String getTitle(){
        return this.title;
    }

    String getID(){
        return this.id;
    }
```

```
    @Override
    public boolean equals(Object obj){
        if(obj == this)
            return true;

        if(obj.getClass() != this.getClass())
            return false;

        final Book book = (Book) obj;
        return this.title.equals(book.title)
            && this.id.equals(book.id);
    }
}

public class BookNotFoundException extends Exception{
    public BookNotFoundException(String errorMsg){
        super(errorMsg);
    }
}

import java.util.Calendar;

class BookStatus {

    private Person person;
    private Calendar due;
```

```
BookStatus() {
    this.person = null;
    this.due = null;
}

BookStatus(Person person, Calendar due) {
    this.person = person;
    this.due = due;
}

void setDueDate(Calendar Cal) {
    this.due = Cal;
}

void setAssociatedPerson(Person p) {
    this.person = p;
}

Calendar getDueDate() {
    return this.due;
}

Person getAssociatedPerson() {
    return this.person;
}
```

```
    boolean isLate() {
        return Calendar.getInstance().after(this.due);
    }
}

import java.util.concurrent.ConcurrentHashMap;
import java.util.Calendar;

class Library {

    private ConcurrentHashMap<Book, BookStatus> library;

    Library() {
        library = new ConcurrentHashMap<Book, BookStatus>();
    }

    Library(ConcurrentHashMap<Book, BookStatus> L) {
        library = L;
    }

    ConcurrentHashMap<Book, BookStatus> getLibrary() {
        return this.library;
    }

    boolean findBook(Book book) {
        BookStatus currentStatus = library.get(book);
        if (currentStatus != null)
```

```
            return true;
        return false;
    }

    boolean findBook(String title, String id) {
        Book book = new Book(title, id);
        BookStatus currentStatus = library.get(book);
        if (currentStatus != null)
            return true;
        return false;
    }

    boolean addBook(Book book) {
        BookStatus currentStatus = library.get(book);
        if (currentStatus == null) {
            currentStatus = new BookStatus();

            BookStatus tempStatus =
                library.putIfAbsent(book, currentStatus);

            if (tempStatus == null) {
                return true;
            } else {
                library.replace(book, tempStatus);
                System.err.println(
                    "The book is already in the library.");
                return false;
```

```
            }
        }
        System.err.println(
            "The book is already in the library.");
        return false;
    }

    boolean removeBook(Book book) {
        BookStatus currentStatus = library.get(book);
        if (currentStatus != null) {

            if (currentStatus.getAssociatedPerson() != null) {
                System.err.println("This book cannot be
                    removed because someone has borrowed it");
                return false;
            }

            BookStatus tempStatus = library.remove(book);
            if (tempStatus != null)
                return true;
        }
        System.err.println("The book is not in the library");
        return false;
    }

    boolean borrowBook(Person person, Book book, int duration)
        throws BookNotFoundException {
```

```
BookStatus currentStatus = library.get(book);
if (currentStatus != null &&
    currentStatus.getAssociatedPerson() == null) {

    if (person.getFine()) {
        System.out.println("The person
            needs to pay the fine first.");
        return false;
    }

    currentStatus.setAssociatedPerson(person);

    Calendar newDate = Calendar.getInstance();
    newDate.add(Calendar.DATE, duration);
    currentStatus.setDueDate(newDate);

    BookStatus tempStatus = library.replace(
        book, currentStatus);
    if (tempStatus.getAssociatedPerson() != null) {
        library.replace(book, tempStatus);
        return false;
    }
    return true;

} else if (currentStatus == null) {
    throw new BookNotFoundException("The book is not found");
```

```
    } else {
        System.err.println(
            "The given book has already been lend out.");
        return false;
    }
}

boolean returnBook(Person person, Book book)
    throws BookNotFoundException {

    BookStatus currentStatus = library.get(book);
    if (currentStatus != null &&
    currentStatus.getAssociatedPerson() != null) {

        if (currentStatus.isLate()) {
            person.setFine(true);
        }

        currentStatus.setAssociatedPerson(null);
        currentStatus.setDueDate(null);

        BookStatus tempStatus =
            library.replace(book, currentStatus);

        if (tempStatus.getAssociatedPerson() == null) {
            library.replace(book, tempStatus);
            return false;
```

```
            }
            return true;

        } else if (currentStatus == null) {
            throw new BookNotFoundException(
                "The book is not found");
        } else {
            System.err.println(
                "The given book has not been borrowed yet.");
            return false;
        }
    }

}

class Person {

    private String id;
    private boolean fine;

    Person() {
        this.id = "";
        this.fine = false;
    }

    Person(String ID, boolean Fine) {
        this.id = ID;
```

```
        this.fine = Fine;
    }

    String getID() {
        return this.id;
    }

    boolean getFine() {
        return this.fine;
    }

    void setID(String ID) {
        this.id = ID;
    }

    void setFine(boolean Fine) {
        this.fine = Fine;
    }

    void payFine() {
        if (this.fine)
            this.fine = false;
    }
}
```

Appendix G

Previous Mental Model Questionnaire

The correct answer to each question is highlighted.

Situation Model Part 1: basic classes

1. Does the program define a "book" class? (Y/N) **Y**
2. Does the program define a "Person" class? (Y/N) **Y**
3. Does the program define a "Collection" class? (Y/N) **N**
4. Does the program define a "Library" that holds the number of copies of a book? (Y/N) **N**
5. Does the program define a class that associates a fine with a person? (Y/N) **Y**
6. Does the program define a class that handles the case when a book is not found? (Y/N) **Y**

Situation Model Part 2: object interactions & data structure

1. Is the map in the Library class a concurrent hashmap? (Y/N) **Y**

2. Does this program check if a person has to pay a fine before he/she borrows any book? (Y/N) **Y**

3. Does this program put a person with a fine into a separate collection if that person returns the book late? (Y/N) **N**

4. Does the "Person" class have a method that removes a fine from a user? (Y/N) **Y**

5. Does this program sort the books according to their title or ID? (Y/N) **N**

6. Does this program check if a person is a student or faculty member? (Y/N) **N**

7. Does the library use book title and book id to find the book? (Y/N) **Y**

8. Is the key type of the hashmap in the "Library" class a string that represents the book title? (Y/N) **N**

9. Is the value type of the hashmap in the "Library" class an instance of "Book-Status"? (Y/N) **Y**

Situation Model Part 3: methods & functionalities

1. Does the method "Map::putIfAbsent()" used in the program return the previous value associated with the given key? (Y/N) **Y**

2. Does the method "Map::remove()" used in this program return the previous value associated with the given key? (Y/N) **Y**

3. Does the method "Map::replace()" used in this program return the current value associated with the given key? (Y/N) **N**

4. Does the method "Map::get()" used in this program use an overridden ".equals()" method to find books in the libraries as keys? (Y/N) **Y**

5. Does the method "Library::addBook()" set a default date (1970/1/1) when a new book is added to the library? (Y/N) **N**

6. Does the method "Library::borrowBook()" allow a book to be lent to other people if someone returns this book late and needs to pay a fine? (Y/N) **Y**

7. Does the method "Library::returnBook()" reset the date to the default date (1970/1/1) when a person returns a book? (Y/N) **N**

8. Is the "Person::payFine()" method associated with any book instance? (Y/N) **N**

Execution Model Part 1: Non-concurrent model

1. Does the program iterate through all the books in the library when someone tries to borrow a book? (Y/N) **N**

2. Does the program modify "BookStatus::due" when the execution of "Library::borrowBook()" returns true? (Y/N) **Y**

3. Does the program modify the "Calendar" value of a "BookStatus" when someone successfully returns the book? (Y/N) **Y**

4. Will the reference of "Person::id" gets changed in the "ConcurrentHashMap" when someone successfully borrows a book? (Y/N) **N**

5. Will the "Person::id" of a person gets changed into the "ConcurrentHashMap" when someone successfully returns a book? (Y/N) **N**

Execution Model Part 2: Concurrent model

1. Does the "addBook" method return "true" on all the threads if two or more threads attempt to add the same book? (Y/N) **N**

2. Does the "removeBook" method return "true" on all the threads if two or more threads tries to remove the same book concurrently? (Y/N) **N**

3. If two threads concurrently execute the "Library::borrowBook()" method, is it possible that one thread returns "true" and the other thread returns "false"? (Y/N) **Y**

4. If two threads concurrently execute the "Library::returnBook()" method, is it possible that one thread returns "true" and the other thread returns "false"? (Y/N) **Y**

5. Does the "Library::findBook()" method return true on all the threads if two or more threads tries to search for the same book, if and only if the book is in the library? (Y/N) **Y**

6. If two threads borrow the same book simultaneously, will one thread modify the book status already modified by another thread? (Y/N) **Y**

7. If two threads return the same book simultaneously, will one thread modify the book status already modified by another thread? (Y/N) **N**

8. Is the program able to identify if another thread has already added a book when the current thread tries to add the same book, assuming both threads calling method "Library::addBook()" and the book is not in the library before the method has been called? (Y/N) **N**

9. Is the program able to identify if another thread has already removed a book when the current thread tries to remove the same book? (Y/N) **Y**

10. Will the "Library::borrowBook()" method throw a "BookNotFoundException" when it tries to borrow a book while another thread removes the book? (Y/N) **Y**

11. Will the "Library::returnBook()" throw a "BookNotFoundException" when it tries to return a book while another thread tries to find the same book? (Y/N)**Y**

12. Is the "ConcurrentHashMap" used in the library shared between threads? (Assuming only one instance of the library is created and the program concurrently runs its methods?) (Y/N) **N**

Appendix H

Previous Demographic Questionnaire

1. Age (in number of years):
2. Are you currently a student? (Yes or No):
3. Number of programming course finished:
4. Number of concurrent programming course finished:
5. Highest level of education achieved:
6. Number of years since you first come to know Java:
7. Number of years of programming experience in Java:
8. Number of years of experience in concurrent programming:
9. Number of years of experience using java.util.concurrent package:
10. Rate your level of expertise in Java concurrency, between 0 and 100 (0 = total lack of knowledge and 100 = total expert):

11. Rate your level of expertise in Java concurrency among peers, between 0 and 100 (0 = total lack of knowledge and 100 = total expert):

12. Email address for the gift card (optional):

Vita

Candidate's full name: Zhangliang Ma

University attended (with dates and degrees obtained): Rensselaer Polytechnic Institute (2016-2020, Bachelor of Cognitive Science)