STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXVIII**, Number 1, 2023
DOI: 10.24193/subbi.2023.1.05

FIELD EXPERIMENT OF THE MEMORY RETENTION OF PROGRAMMERS REGARDING SOURCE CODE

ANETT FEKETE AND ZOLTÁN PORKOLÁB

Abstract. Program comprehension is a continuously important topic in computer science since the spread of personal computers, and several program comprehension models have been identified as possible directions of active code comprehension. There has been little research on how much programmers remember the code they have once written. We conducted two experiments with a group of Computer Science MSc students. In the first experiment, we examined the code comprehension strategies of the participants. The students were given a task to implement a minor feature in a relatively small C++ project. In the second experiment, we asked the students 2 months later to complete the same task again. Before starting the clock, we asked the students to fill a questionnaire which aimed to measure program code-related memory retention: we inquired about how much the students remembered the code, down to the smallest relevant details, e.g. the name of functions and variables they had to find to complete the task.

After the second experiment, we could compare the solution times of those students who participated in both parts. As one result, we could see that these students could solve the task in shorter time than they did in the first experiment. We also looked at the results of the questionnaire: the vast majority of students could not precisely remember more than two or three identifiers from the original code. In this paper, we will show how this result compares to the forgetting curve.

1. Introduction

Software development is a knowledge-intensive and complex task that demands programmers to master and utilize vast amounts of information. Programmers need to have a deep understanding of programming languages, algorithms, and design patterns, among other things. Moreover, the retention of knowledge and memory of previously written and read code is essential for

Received by the editors: 1 March 2023.

2010 *Mathematics Subject Classification.* 68U99.

1998 *CR Categories and Descriptors.* I.m [**Computing Methodologies**]: Miscellaneous; J.m [**Computer Applications**]: Miscellaneous .

Key words and phrases. code comprehension, memory retention, experiment.

program comprehension and efficient coding. The memory retention of coding concepts is important for programmers' productivity, as they must be able to recall previous code when creating new programs.

The ability to remember code and programming concepts is a critical component in the development process, but it is not always clear how long the retention lasts, or how it impacts performance. To address these questions, this paper investigates the effect of forgetting on source code comprehension and task solving time. We also examine whether programmers tend to remember code details or larger units, such as functions or algorithms. By answering these questions, we can gain a better understanding of the cognitive processes involved in programming and provide insights into how programmers can optimize their performance by retaining and recalling code more effectively.

In order to measure memory retention regarding source code, we conducted two experiments with Computer Science MSc students, in which the participants had to solve the same programming task and answer memory-related questions. The experiments took place two months apart. After the second experiment, we investigated the memories the participants had of the task, and how that influenced their solution time. We asked the students to describe their memories with as much details as possible in an essay question, and asked them to fill a multiple-choice question which targeted the remembrance of exact source code details.

In this paper, we attempt to answer the following research questions in connection with source code comprehension and memory retention:

- **RQ1:** How does forgetting affect source code comprehension and task solving time?
- **RQ2:** Are programmers more likely to remember the details of the code, or larger units like functions or algorithms?

The rest of the paper is structured as follows: In Section 2, we present earlier research about memory retention and programming experiments. In Section 3, we describe the details of both experiments, putting more focus on the second one. Section 4 contains the results of the second experiment. In Section 5, we mention the possible threats to the validity of our study. Finally, we conclude the paper in Section 6.

2. Related work

Our work is focused on the memory retention of programmers regarding source code through two experiments in which the participants were given a programming task to solve. In this section we present related research to show how other studies conducted experiments that were centered around the work of software developers, and attempts to measure how programmers remember source code.

2.1. **Program comprehension experiments.** Programming tasks require cognitive effort and mental models from programmers, and can affect them physically. Many experiments have investigated program comprehension and computer science students, examining various aspects. For example, Nakagawa et al. measured cerebral blood flow during program comprehension and found that more complex code increased mental workload [14]. Andrzejewska and Skawińska tracked eye activity and found that external conditions and cognitive load affected comprehension speed [1]. Krüger et al. examined feature traceability and program decomposition and found that feature traces were helpful in solving tasks more quickly, while program decomposition hindered it [10]. They also found based on developer interviews that self-assessments are reliable sources of developer-related information, and programmers tend to be correct when they recall memories on project-related questions that they consider important [11]. Their findings confirm the study of Koenemann and Robertson who investigated the analysis methods of professional developers, and found that they focus on the software parts that they perceive as relevant to them [9]. Cornelissen, Zaidman, and Dursen investigated trace visualization and found that it could speed up task solving by 22% [3]. Kather and Jan found that program comprehension and algorithm comprehension are not the same, and that domain knowledge, experience, and abstract knowledge can help solve tasks more quickly [8].

2.2. **Memory retention and forgetting.** In 1885, Ebbinghaus defined the so-called ”forgetting curve” [4] (see Figure 1) after a series of experiments in which his subjects tried to remember randomly selected words. The most important factor of the formula is time. The original experiments of Ebbinghaus were since then replicated, confirming the correctness of the formula with some small modification in its smoothness [13], and it has also been investigated in the context of brain function [16]. The psychological experiments of Averell and Heathcote [2] confirmed that the exponential curve is the best fit to model human forgetting.

Forgetting and memory retention has been scarcely researched from a software development and source code aspect. Some studies that utilize the forgetting curve include the work of Xu et al. who investigated the concreteness and readability of identifiers in the source code based on how easily programmers remembered them [18]. One study that is closer to our goals is the work of Ünal et al. who looked at how repeated exposure to the same source code helps solving programming tasks [17]. Kang and Hahn found in their study

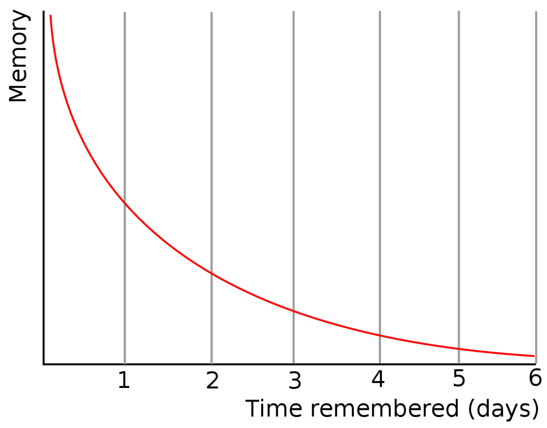

Figure 1. The forgetting curve as described by Ebbinghaus in the 19th century.

that forgetting affects methodological knowledge more than technology-related knowledge [7]. Most similarly to our research objectives, Krüger et al. examined whether the forgetting curve is applicable in remembering source code. Their experiment covered hours and days during which the programmers were asked to recall memories of the source code [12]. Our aim is to investigate whether programmers are likely to remember abstract levels and details of the source code after a longer time period.

3. Experiments

We planned two experiments in advance: in the first one, we targeted the code comprehension strategies of junior programmers. The goal of the second experiment was to gain an understanding of the memory retention of programmers of source code.

We asked Computer Science MSc students from Eötvös Loránd University to take part in the experiment. The students were all enrolled in the *Multi-paradigm programming* course whose main topic is advanced C++. A total of 27 students took part in the first experiment, and 16 of them took part in both. We considered MSc students to be better experimental subjects, since they generally have more experience in programming (both as a job and as an activity), and because of that, they are more conscious about code comprehension and programming tasks.

Both experiments consisted of two main parts: first, the students were asked to fill a different questionnaire. Afterwards, the students were given a small

C++ task to solve in *TinyXML2*[1], a simple XML parser which contains only three C++ source files of hundreds of lines of code. The task was focused on code comprehension rather than writing new code: TinyXML2 is case-sensitive by default regarding XML tags. The students had to make the library case-insensitive by finding a particular line of code, and replace it with calling a function that we readily provided for them. Thus, we could measure code comprehension speed because the students only had to focus on understanding the code and finding the line in question instead of spending time with writing the replacement code.

The following function definition was provided for the participants:

```
#include <ctype.h>

int my_stricmp(const char* s1, const char* s2)
{
    while (tolower((unsigned char) *s1) == tolower((unsigned
        ↪ char) *s2))
    {
        if (*s1 == '\0')
            return 0;
        s1++; s2++;
    }
    return (int) tolower((unsigned char)*s1) - (int) tolower
        ↪ ((unsigned char)*s2);
}
```

The line to be modified was line 1142 in *tinyxml.cpp*:

```
...
else if ( !XMLUtil::StringEqual( endTag.GetStr(), ele->Name
    ↪ () ) ) {
...
```

The correct solution:

```
...
else if ( my_stricmp( endTag.GetStr(), ele->Name() ) ) {
...
```

We divided the students into two groups: one group had to use CodeCompass for code comprehension activities, the other group was free to use any code editor or code comprehension tool. The latter group formed the control

[1]TinyXML2 GitHub repo: `https://github.com/leethomason/tinyxml2`

group in the first experiment. CodeCompass [15] is an open-source code comprehension framework which applies static analysis to the source code and its environment (e.g. compilation database, version control repository), and provides various textual and visual support for understanding source code both on code level and file level.

3.1. **First experiment.** In the first experiment, we investigated the usual code comprehension strategies of young programmers, and how that correlates with factors such as the amount of experience as a programmer, and language familiarity. Building on our earlier study [6], we aimed to investigate the comprehension functionality that students used during task solution.

In the questionnaire of the first experiment we inquired about the amount of their work and general programming experience, as well as the languages they were most familiar with. As mentioned above, 27 students took part in this experiment: 15 had to use CodeCompass, and 12 were free to use any other tool.

Based on the participants' solution time and their answers to the questionnaire, we concluded that while more programming experience meant quicker task solution, work experience correlated more with solution time. The students in the CodeCompass group used our demo server[2], which collects anonymous user activity using Google Analytics. The activity log in CodeCompass showed that the students were majorly using top-down comprehension strategies.

The details and results of the first experiment are elaborated in our previous study [5].

3.2. **Second experiment.** The second experiment took part cc. two months after the first one. As mentioned above, 16 students took part in both the first and the second experiment, thus their results are relevant in this study.

The students were asked to solve the same programming task as in the first experiment: find the line of code in TinyXML2 in which the function call needs to be replaced with the provided function in order to make XML parsing case-insensitive.

We asked them to fill a different questionnaire the second time. The questions were related to source code memory retention:

- Essay question: *What do you remember from the first experiment? Please provide as much information as you can, any detail can be useful.*
- Multiple-choice question: *Which identifiers were in the program that you had to modify?* For this question we listed 9 correct and 21 false

[2]Demo server: `https://codecompass.net/`

identifiers. The false ones were most of the time very similar to the correct ones, or they were made to sound relevant in an XML parser.

All students had to use the same tool for comprehension activities they used in the first experiment. Our goal was to repeat the first experiment down to every possible detail, in order to remove any additional factors that might affect measuring memory retention. Both experiments were conducted in the same university computer lab, and the machines were equipped with the same hardware and software.

4. Results and discussion

By repeating the experiment, and asking the students about their memories of the first experiment, we wanted to investigate whether participants remember source code details or structure, and whether remembering details of the actual code correctly is correlated with quicker solution time. In our previous study, we collected the common elements of code comprehension models [6]. These elements usually rely on how the code is written syntactically (e.g. beacons are ”visual cues” in the code the programmer is looking for to identify the meaning of a source code unit), this is why we focus on remembering actual identifiers.

We evaluated the students’ responses to the essay question. 14 out of 16 students remembered the task clearly, and 9 students described steps of their previous solution. It is worth noting, that multiple students explicitly stated in their response that they do not usually remember exact identifiers of any source code, instead they remember structural details.

Table 2 shows the number of correct and incorrect guesses of the multiple-choice question for each student. We calculated the χ-square test for the answers of the question to determine whether the distribution of responses is significantly different from what would be expected by chance. Table 1 represents the contingency table of the calculation. We divided the responses into two categories, marked and not marked.

Equation 1 shows the results of the test. The degree of freedom in the calculation was 1, and the original significance was $p < .05$. The statistic value and the calculated significance suggest that the students’ guesses were influenced by the correct vs. incorrect nature of the answer. Equation 2 shows the statistics of the test with Yates correction: the results in this case did not change the conclusion, the null hypothesis (that the students’ answers are independent of correctness of the answer) remains rejected.

$$\chi^2(1) = 21.42, p < .00001 \qquad (1)$$

	Correct answers	**Wrong answers**
Marked	58	68
Not marked	77	247

Table 1. The contingency table used in the χ-square test for the evaluation of the multiple-choice question in which we examined if the students remember identifiers correctly.

$$\chi^2(1) = 20.37, p < .00001 \tag{2}$$

In Table 2 we also listed the solution times of each student who took part in both experiments. Comparing the two experiments, the solution times show an average improvement of 16 minutes and 20 seconds. If we look at the individual solution times, we can see that students performed better in all cases we knew both solution times.

The proportion of correct vs. wrong guesses was greater or equal to 1 in the case of 8 students, while this number was below 1 for the other 8 students. Comparing the solution times, the students with better guess rate improved by 16.97 minutes on average, while the other 8 students decreased their average solution time by 15.6 minutes. The cc. 1.5-minute difference between the average improvements shows that remembering identifiers better correlates with quicker solution time. However, the significant improvement in solution times for all participants suggests that remembering the process of task solution is more significant than remembering exact identifiers in the source code. We included in the rightmost column of Table 2 if a student described steps or details of the solution in the essay question. 7 students who had more correct than wrong guesses reported such memories, while only 2 students remembered any details from the solution. This result suggests that remembering steps of an algorithm and exact details from a code base are correlated.

To answer **RQ1** *(How does forgetting affect source code comprehension and task solving time?)*, our data shows that the participants who reported more memories of the first experiment - either in the form of actual identifiers or verbal descriptions of the task or the source code - performed better on average during the second experiment.

Solution times and responses to the questions suggest that the participants had statistically significant memories of the source code after two months of the initial experiment. To answer **RQ2** *(Are programmers more likely to remember the details of the code, or larger units like functions or algorithms?)*, the data suggests that there is a correlation between remembering exact details of the source code and having more memories of the structure or steps of solving a programming task.

The findings in reply to RQ2 are complementary to a related study [11] that concluded from developer interviews that abstract knowledge of the source code is more important to remember. However, our results are somewhat contradictory of another study by Krüger et al. [12] who found that the forgetting curve applies to remembering source code. According to our results, the participants had a fairly good recollection of the solution process even after 2 months. This suggests that forgetting slows down after a certain amount of time, as we observed memory retention after two months, and the aforementioned study investigates remembering source code after some days.

Student #	**Solution time #1 (mins)**	**Solution time #2 (mins)**	**Correct ids**	**Wrong ids**	**Detailed memories?**
1	**11**	**3:05**	**7**	**7**	✓
2	N/A	25	5	7	✗
3	**33**	**8:03**	**3**	**2**	✓
4	19:50	N/A	2	2	✗
5	**N/A**	**14**	**6**	**4**	✗
6	11:30	8:54	5	6	✓
7	26:40	4:34	1	3	✓
8	**30**	**8**	**7**	**3**	✓
9	**7**	**1:20**	**4**	**4**	✓
10	23:48	16:54	2	7	✗
11	37	27	1	3	✗
12	59:48	19	2	3	✗
13	**24**	**2:50**	4	**3**	✓
14	**N/A**	**N/A**	**7**	**0**	✓
15	**12**	**4:30**	4	**2**	✓
16	26:47	15:35	5	12	✗

Table 2. The results of the second experiment. Comparing solution times we can see that all students performed better the second time which implies complex memory retention in spite of inconsistent remembrance of exact identifiers.

5. Threats to validity

As any research that relies on human resources and input, our study holds some obvious threats to validity.

Small number of participants. Although 27 students took part in the first experiment, only 16 of them was present during the second one. The students come from similar backgrounds considering their computer science education and programming experience. This might narrow down our research results regarding target population, making more experiments needed with a more diverse pool of participants.

Incomplete data. The questionnaire was available for the students on the Canvas learning management system. The responses of one student could not be found after the experiment. The solution times were also collected through Canvas, and some data was lost between the experiments, this is why a few solution times are missing from Table 2. The missing data is omitted in our calculations in order to avoid distorting results, hence the metrics in our results are computed for 12 students instead of 16, the total number of participants.

Short study period for the students. In our experiments, the students had one hour both times to study the source code of TinyXML2. In reality, a programmer spends much more time working on the same source code, so their memory retention of the code is probably stronger. However, our data shows that even with a short study period and after a longer intermission, the students were able to recall the comprehension process and solved the task quicker than the first time, which suggests that more time spent with the same code instills even stronger memories.

Effects of the first experiment. The students were aware that they were participating in an experiment both times which gives space to biased results as they might have paid more attention to the code and exercise than they would have had they not know about the experiment. However, at the time of the first experiment, they did not know there would be a second one so they had no direct reason to clearly remember the details of the first one after the 2-month break.

6. Conclusion

In this research, we presented the results of two consecutive experiments with Computer Science MSc students in which we investigated the effect of forgetting in source code comprehension and solving programming tasks. The students were asked to fill questionnaires and solve the same C++ programming task in both experiments. In the first experiment, we examined the code comprehension strategies of the students, and the correlation between task solution time, and work experience, general programming experience, and familiarity with programming languages. In the second experiment, we investigated how much the students remembered from the first experiment: we asked them to describe their memories, and answer a multiple-choice question about the actual identifiers in the source code.

In total, 16 students took part in both experiments. We executed a χ-square test on the students' guesses in the multiple-choice question. The test showed that there is correlation between the correctness of an answer option and if it was guessed by a student.

The average solution time was decreased by 16.3 minutes on average among the participants. In case of the 8 students who marked at least as many correct identifiers as wrong ones in the multiple-choice question, the solution time improved by 15.6 minutes, and 16.97 minutes for the other 8 students. This result suggests that remembering the process of task solution is a more significant factor in code comprehension than remembering exact identifiers. The results of our research suggest that remembering both structural and code-level details contribute to quicker task solution, and that remembering more exact details of the source code correlates with the retention of more structural memories.

References

1. Magdalena Andrzejewska and Agnieszka Skawińska, *Examining students' intrinsic cognitive load during program comprehension–an eye tracking approach*, International Conference on Artificial Intelligence in Education, Springer, 2020, pp. 25–30.
2. Lee Averell and Andrew Heathcote, *The form of the forgetting curve and the fate of memories*, Journal of mathematical psychology **55** (2011), no. 1, 25–35.
3. Bas Cornelissen, Andy Zaidman, Arie Van Deursen, and Bart Van Rompaey, *Trace visualization for program comprehension: A controlled experiment*, 2009 IEEE 17th International Conference on Program Comprehension, IEEE, 2009, pp. 100–109.
4. Hermann Ebbinghaus, *Über das gedachtnis*, 1885.
5. Anett Fekete and Zoltán Porkoláb, *Report on a Field Experiment of the Comprehension Strategies of Computer Science MSc Students*, 2022 IEEE 16th International Scientific Conference on Informatics - Proceedings, IEEE, 2022, pp. 73–81.
6. Anett Fekete and Zoltán Porkoláb, *A comprehensive review on software comprehension models*, Annales Mathematicae et Informaticae, vol. 51, Líceum University Press, 2020, pp. 103–111.
7. Keumseok Kang and Jungpil Hahn, *Learning and forgetting curves in software development: Does type of knowledge matter?*, ICIS 2009 Proceedings (2009), 194.
8. Philipp Kather and Jan Vahrenhold, *Is algorithm comprehension different from program comprehension?*, 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), IEEE, 2021, pp. 455–466.
9. Jürgen Koenemann and Scott P Robertson, *Expert problem solving strategies for program comprehension*, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 1991, pp. 125–130.
10. Jacob Krüger, Gül Çalıklı, Thorsten Berger, Thomas Leich, and Gunter Saake, *Effects of explicit feature traceability on program comprehension*, Proceedings of the 2019 27th

ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2019, pp. 338–349.

11. Jacob Krüger and Regina Hebig, *What developers (care to) recall: An interview survey on smaller systems*, 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2020, pp. 46–57.
12. Jacob Krüger, Jens Wiemann, Wolfram Fenske, Gunter Saake, and Thomas Leich, *Do you remember this source code?*, Proceedings of the 40th International Conference on Software Engineering, 2018, pp. 764–775.
13. Jaap MJ Murre and Joeri Dros, *Replication and analysis of ebbinghaus' forgetting curve*, PloS one **10** (2015), no. 7.
14. Takao Nakagawa, Yasutaka Kamei, Hidetake Uwano, Akito Monden, Kenichi Matsumoto, and Daniel M German, *Quantifying programmers' mental workload during program comprehension based on cerebral blood flow measurement: a controlled experiment*, Companion proceedings of the 36th international conference on software engineering, 2014, pp. 448–451.
15. Zoltán Porkoláb, Tibor Brunner, Dániel Krupp, and Márton Csordás, *Codecompass: an open software comprehension framework for industrial usage*, Proceedings of the 26th Conference on Program Comprehension, 2018, pp. 361–369.
16. Dong Gue Roe, Seongchan Kim, Yoon Young Choi, Hwije Woo, Moon Sung Kang, Young Jae Song, Jong-Hyun Ahn, Yoonmyung Lee, and Jeong Ho Cho, *Biologically plausible artificial synaptic array: Replicating ebbinghaus' memory curve with selective attention*, Advanced Materials **33** (2021), no. 14, 2007782.
17. Utku Ünal, Eray Tüzün, Tamer Gezici, and Ausaf Ahmed Farooqui, *Investigating the impact of forgetting in software development*, arXiv preprint arXiv:2204.07669 (2022).
18. Weifeng Xu, Dianxiang Xu, and Lin Deng, *Measurement of source code readability using word concreteness and memory retention of variable names*, 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), vol. 1, IEEE, 2017, pp. 33–38.

Eötvös Loránd University,, Faculty of Informatics, Egyetem tér 1-3.,, 1053 Budapest,, Hungary
Email address: `afekete@inf.elte.hu`

Eötvös Loránd University,, Faculty of Informatics, Egyetem tér 1-3.,, 1053 Budapest,, Hungary
Email address: `gsd@inf.elte.hu`