Understanding the Differences Between Novice and Expert Programmers in Memorizing Source Code

Matthias Kramer, Mike Barkmin, David Tobinski, Torsten Brinda

Understanding the Differences Between Novice and Expert Programmers in Memorizing Source Code

Matthias Kramer[1], Mike Barkmin[1], David Tobinski[2] and Torsten Brinda[1]

[1] University of Duisburg-Essen, Didactics of Informatics, Essen, Germany
[2] University of Duisburg-Essen, Cognitive and Educational Psychology, Essen, Germany
{matthias.kramer, mike.barkmin, torsten.brinda}@uni-due.de
david.tobinski@uni-due.de

Abstract. This study investigates the difference between novice and expert programmers in memorizing source code. The categorization was based on a questionnaire, which measured the self-estimated programming experience. An instrument for assessing the ability to memorize source code was developed. Also, well-known cognitive tests for measuring working memory capacity and attention were used, based on the work of Kellog and Hayes. Forty-two participants transcribed items which were hidden initially but could be revealed by the participants at will. We recorded all keystrokes, counted the lookups and measured the lookup time. The results suggest that experts could memorize more source code at once, because they used fewer lookups and less lookup time. By investigating the items in more detail, we found that it is possible that experts memorize short source codes in semantic entities, whereas novice programmers memorize them line by line. Because our experts were significantly better in the performed memory capacity tests, our findings must be viewed with caution. Therefore, there is a definite need to investigate the correlation between working memory and self-estimated programming experience.

Keywords. Assessment; Object-oriented programming; Working memory; Programming experience

1 Introduction

The identification and empirical validation of competency structures have been one of the core topics in German educational research during the last few years, see [1,2]. Fueled in addition by the results of international comparative studies such as PISA[1], TIMSS and PIRLS[2] the German educational system had to undergo a rigorous restructuring process [3] from an input oriented and teacher centered system to an output oriented and learner centered system. Instead of concentrating on the specific content to teach, the focus shifted to the skills and abilities that learners need to solve problems in a specific situation. These skills and abilities are typically described with

[1] See https://www.oecd.org/pisa/
[2] See http://timss.bc.edu/

the term *competency*. A precise definition was given by Weinert and can be found in [4].

During this shift, a priority programme[3] of the German national science foundation, the DFG, was initiated. The research areas covered several educational fields and ranged from the theoretical derivation of competencies and their gathering in competency models up to practical implications for people who work in the educational field [5]. The approaches in the numerous sub-projects were the basis for our project COMMOOP which deals with the determination and empirical validation of competencies from beginners in the area of object-oriented programming (OOP). The process of literature-based derivation as well as a first resulting version of the competency model is documented in [6]. First assessment results for the competency facets of recognizing object-oriented syntax elements in given source code can be found in [7].

These results already gave first hints, that beginners identify syntax elements based on taught code conventions, such as position of elements in the code or upper/lower case of letters but not on a semantic level. Based on these results and referring to the results of Adelson [8] we assume that programming experts have internalized the programming syntax. Hence, we hypothesize that they mentally compile given code structures into semantic entities and memorize the working algorithm behind while beginners tend to memorize syntactic elements. To confirm this hypothesis, we conducted a test where we asked participants to reproduce given code. Furthermore, we investigated the influence of the working memory to examine the contributions of "natural" memorization abilities. Referring to our competency model, this test assesses competency facets *Syntax and Semantics* from the competency dimension *Mastering Representation* in conjunction with the cognitive process of *Remembering*. In Section 2 we give an overview on the theoretical work on the concept of working memory in the field of cognitive psychology as well as research results with similar settings. Section 3 includes the description of the test instrument and the items included there. A presentation of the results of a first test is given in Section 4, followed by a discussion in Section 5 and an outlook for further work in Section 6.

2 Background and Related Work

Basic processes such as editing a natural language text may require some cognitive systems. For example, Kellog [9] assumed, that specific parts of the working memory are used in such processes. Hayes [10] independently proposed a broader model of the role of working memory in writing natural language texts, which also shows the interaction between an individual and a task environment. He divided the task environment into two interacting components – the social environment and the physical environment. The individual was categorized into motivation, cognitive processes, long-term memory and working memory, which also interact with each other. Although programming languages are formal languages, the model gave us an indication on which interactions could happen during the process of writing source code. Because we used this model as a reference for a transcription task, which only

[3] See http://kompetenzmodelle.dipf.de/en/

involved reproducing source code, the task environment and the motivation/affect component were not relevant for our study and hence we reduced the model by taking only the individual component in account.

Therefore, we reduced the model, by only considering long-term memory, working memory and cognitive processes (see Fig. 1) and translated the components. The cognitive process “text interpretation” was translated to “code analyzing” and “text production” to “code production”. Long-term memory components were replaced with “OOP knowledge and skills” and “mastering representations” as proposed in our competency structure model [6]. Due to our aforementioned interest in code reproduction, we were specifically interested in assessing the participant’s working memory. Hayes derived the working memory component from Baddeley's and Hitch's model [11], which consists of the following four parts:

1. phonological loop/memory (stores phonological information, such as a telephone number, and prevents its decay by continuously refreshing it in a rehearsal loop)
2. visuo-spatial (visual/spatial) sketchpad (stores visual and spatial information, such as the arrangement of chairs in a room)
3. semantic memory/episodic buffer (contains information that combine phonological, visual, and spatial information)
4. central executive (controls the attention and therefore filters unnecessary information and coordinates cognitive tasks)

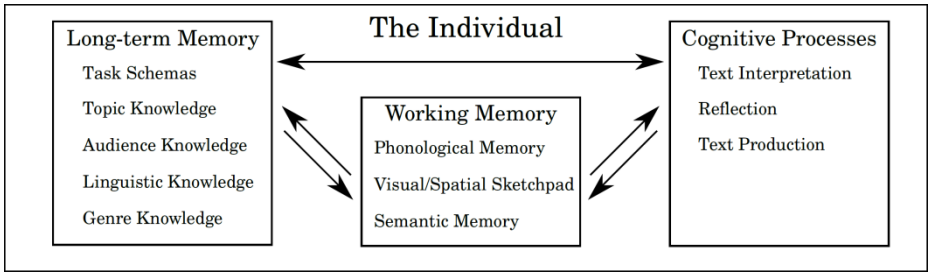

Fig. 1. Reduced Hayes-Flower-Model (Source: [10], p.4)

In the last decades, there have been several studies to show the interaction of these components in various contexts. In 1965, de Groot [12] conducted a study with master chess players, which proved that they only need five seconds to study a midgame board to reproduce it with 90% accuracy. Chase and Simon [13] reproduced the study and found that, master chess players memorized attacking and defending formations, rather than individual pieces. Thus, they could reproduce the board with a much higher accuracy than normal chess players. Adelson [7] used the results of de Groot and Chase and Simon to test if this phenomenon occurs in programming related tasks as well. In Adelson's experiment novice and expert programmers were shown sixteen randomly ordered lines of Polymorphic Programming Language source code, which they should remember and later recall. By testing five novices and five experts, Adelson found that experts are more likely to remember a higher number of source code lines. This study also investigated why experts are more capable of memorizing the source code lines. Adelson found, that novice programmers tried to memorize the

source code in syntactic categories. Experts, however, tried to memorize it in semantic categories. So, Adelson concluded that with more expertise, the categories, which are used to remember source code, are getting more complex.

This paper attempts to combine Adelson's and de Groot's ideas by changing the amount of code given at a time. We wanted to find out, if the same behavior as described by Adelson occurs, when the subjects get to view the full source code at once like de Groot's experiment. Moreover, our purpose was to extend the existing research by taking a psychological perspective into regard. We therefore designed a study in which the subjects do not only run through programming tasks but also through tests from the domain of the cognitive psychology.

3 Methodology

In this study, most of the participants were tested using well-known cognitive tests to measure the attention ability and working memory capacity. They also transcribed natural language texts and source codes to tests if there is a difference between expert and novice programmers. The complete test design is described in Section 3.1, followed by an explanation on how we constructed the items in Section 3.2 and an overview of our participants is given in Section 3.3.

3.1 Design

To identify a correlation between the performance in memorizing and reproducing source code and programming experience, we proceeded in two steps.

In the first step, we gathered four well-reviewed cognitive tests to measure the attention ability and working memory capacity of the subjects. One of the most prominent tools for assessing the visual-spatial sketchpad is the Corsi block-tapping-test [14]. It is originally executed on a wooden board with blocks on it, but for our study we used a computer-based version to better control the experimenter artifact. For relating our results with the ones of Brunetti et al. [15], who already developed and tested an e-Corsi test in 2014, we took their description as a draft for our implementation. For further measuring of the working memory, we used two modules of the intelligence structure test I-S-T 2000R [16]. One of them is used to measure the ability to memorize verbal expression (capacity of the phonological loop), whereas the other focused on figural objects (capacity of the visuo-spatial sketchpad). To quantify the attention of the subjects (central executive), we used the d2 test of attention [17], which measures the selective and sustained attention and visual scanning speed.

In the second step, we tested the ability to memorize and reproduce source code. For that purpose, we asked the subjects to transcribe text. First, the content was presented on the left side of the screen and should be transcribed to a text area on the right side of the same screen. This was used to calculate the typing speed (keystrokes per seconds) of the subjects. Afterwards, the content was hidden but could be revealed by the participants on their own demand. To prevent the subjects from cheating by taking notes or a picture, the text was only revealed if the keys <CTRL> and <ALT> and a mouse button were pressed. A pre-test has shown, that we cannot require a 100 percent match, because in some cases misspellings or transpositions of two nearby

characters were hard to find and have resulted in a much longer time required to process an item. Therefore, we used the Levenshtein-distance [18] to determine how similar a subject's text and the correct one was. The algorithm calculates the minimum distance between two strings. The minimum distance is defined as the smallest number of deletions, insertions and reversals that will transform a string A into a string B. After another test, we found, that 95 percent similarity (one minus current Levenshtein-distance divided by Levenshtein-distance with an empty string) was a good threshold. The subjects were informed on the current similarity of their text with the correct one, by showing a process indicator above the text area. It changed its color to green, when the threshold was reached to indicate that the subject can submit his/her work. During each keystroke, we have saved the respective key, the time and the Levenshtein-distance. We also saved each period when the hidden text was revealed and concealed again. Keeping in mind this format could have been unknown to the subjects, we implemented test items for the hidden and for the visible content tasks.

The pre-tests already gave us the hint that both parts of the test (assessment of the several parts of the working memory as well as reproducing source code) could respectively be finished in about 30 minutes, so that the whole test didn't take up more time than 60 minutes altogether. After these tests were completed, the subjects were asked to give further information on their programming experience in a questionnaire. Five items were constructed using the results of Siegmund et al. [19], who researched the correlation between questionnaires used in other researches and performance in solving program-comprehension tasks. They have found that self-estimation indicates programming experience well.

3.2 Item construction

For the hidden content task three groups of three items were constructed. Each group contained a source code with a class structure, a source code with an algorithm and a natural text. These should be comparable to each other in the dimension of character count and complexity. For the items of group 1 see Fig. 2.

```
public class Haus {
  private int nummer;
  private String farbe;

  public void
   streiche(String farbe) {
    this.farbe = farbe;
  }
}
```

```
Lettland ist ein
Staat im Osten von
Europa. Er liegt an
der Ostsee und
gehört zu den
baltischen Staaten.
Die beiden anderen
sind Estland und
Litauen.
```

```
boolean istVielfaches(int
    zahl, int vielfaches) {
  if(vielfaches % zahl ==
       0){
    return true;
  } else {
    return false;
  }
}
```

Fig. 2. Item Group 1

Items of group 2 consists of a class "Datei" composed of two attributes, a constructor and a setter, a text about "Lettland" and an algorithm "einruecken" to indent a given string by using a symbol. The last group contained an item with a class "Vieleck" and a subclass "Dreieck", a text about "Allerheiligen" and an algorithm "istPrimzahl" to test if a number is prime. Therefore, we calculated the complexity of the classes with the weighted class complexity (WCC) by Misra and Adewumi [20]. We have chosen this complexity metric, because it uses a cognitive weight for basic

control structures to calculate a method complexity (MC), which we used for the algorithms.

The complexity of the texts was calculated by using a readability index, which indicates how easy it is to read the text. It should be mentioned, that understanding the text is not considered. Because the readability is dependent on the grammar of the language, we used special readability indexes for German, namely the Flesh-Reading-Ease for German (FRE) [21], based on the original Flesh-Reading-Ease [22], which calculates the readability in values between 0 and 100 (lower = more difficult to read), and the "Wiener-Sachtext-Formel" (WSF) [23], which calculates the readability in school years between 4 and 15 (higher = more difficult to read). We took these two, because they make slightly different assumptions of what is difficult to read. By respecting both we can argue, that our chosen texts got increasingly more complex to read in every aspect.

These considerations resulted in three groups. The first group contains items with the lowest character count and complexity. The last group contains items with the highest character count and complexity compared to the others.

Table 1. Item Groups

Group	Class		Text			Algorithm	
	Chars	WCC	Chars	FRE	WSF	Chars	MC
Group 1	147	3	149	79	5	146	2
Group 2	219	5	206	69	8	214	4
Group 3	237	7	221	49	10	244	8

For the visible content task, we constructed two items, one text and one algorithm, which were comparable in character count and complexity to the items of group 3.

3.3 Participants

Forty-two students were recruited for this study. Thirty of them studied in a field related to computer science and had at least finished an introduction course for object-oriented programming. Of this group, six had a bachelor's degree and two a master's degrees. The remaining students studied something unrelated to computer science. One third of the sample size were female.

4 Results

We have used the recorded keystrokes and lookup times to analyze the difference between novice and experts for each item. Therefore, we have divided the group into novice and experts by using the median of the sum of the programming experience items of the questionnaire. This is allowed, because the internal consistency of the five items was excellent (cronbach's alpha = 0.96). The maximum score possible was 50, the lowest 5 and the median 29.

The typing speed and error rate (percentage of <Del> and <Backspace>) of the subjects showed no significant correlation with the programming experience. The performance of the subjects in the Corsi block-tapping-test did not significantly

correlated with our measures. The d2 test only correlated with the last item of our test (r = 0.37, p-value = 0.0248).

4.1 Lookup Count

We found, that a higher score in the working memory test correlated with the difference in lookup count in many of our items (see Table 2). Also, the programming experience correlated with all items expect Class2. Therefore, we tested if we could find group differences between novice and experts by executing a Mann-Whitney U test analysis. We used this test, because our data was not normally distributed which we found out by using the Kolmogorow-Smirnow-test.

Table 2. Correlation between lookup count, working memory and programming experience (IST-F = I-S-T 2000R figural, IST-V = I-S-T 2000R verbal, PE = programming experience)

	Class1	Algo1	Class2	Algo2	Class3	Algo3
IST-F	0.42*	0.46**	0.13	0.10	0.47**	0.39*
IST-V	0.47**	0.36*	-0.01	0.25	0.48**	0.69***
PE	0.52***	0.73***	0.28	0.22	0.33*	0.43*

Note: * p < .05; ** p < .01; *** p < .001

The U test analysis showed, that the difference in lookup count between the normal text and the source codes was significantly higher for the novice than for the experts in the items of the first group (class: W = 91, p-value = 0.0015, algorithm: W = 61, p-value = 0.001). The mean lookup count of novices was 2.47 units higher for the class item and 3.24 units higher for the algorithm one than for the lookup count of the first normal text. Experts had a mean lookup count difference of -0.40 for the class item and 0.40 for the algorithm one. The lookup count difference in class item of the third group was significantly (W = 104, p-value = 0.045) higher for the novice (4.24) then for the experts (2.4). The algorithmic item and second group showed no significance (p-values between 0.27 and 0.75) in this regard.

4.2 Lookup Time

The programming experience correlated significantly with Class1, Algo1, Class2, Algo3. The verbal working memory test only correlated with Class2 and the figural working memory test only correlated with Algo3 (see Table 3).

Table 3. Correlation between lookup time, working memory and programming experience (IST-F = I-S-T 2000R figural, IST-V = I-S-T 2000R verbal, PE = programming experience)

	Class1	Algo1	Class2	Algo2	Class3	Algo3
IST-F	0.40*	0.43**	0.18	-0.03	0.27	0.38*
IST-V	0.15	0.20	0.07	0.03	0.11	0.41*
PE	0.53***	0.71***	0.39*	0.24	0.22	0.38*

Note: * p < .05; ** p < .01; *** p < .001

Table 4 shows the results of the U test analysis. There was a significant difference between experts and novices regarding the lookup time difference in items Class1, Algo1 and Class3.

Table 4. U test results. Lookup time difference between experts and novices.

	Class1	Algo1	Class2	Algo2	Class3	Algo3
Experts	2.01	3.05	8.64	-7.54	-4.64	-8.58
Novices	-6.30	-7.89	0.30	-12.83	-9.60	-18.50
W	73***	55***	116	140	96*	120

Note: * $p < .05$; ** $p < .01$; *** $p < .001$

4.3 Working Memory

Because general working memory capacity could influence our results, we also tested on group difference in regard to the I-S-T 2000R verbal and figural score. Only in figural working memory we could find a significant difference (IST-V: W = 132, p-value = 0.212, IST-F: W = 81, p-value = 0.006), hence our experts had a better working memory capacity in this regard.

5 Discussion

It was hypothesized that expert programmers would memorize the working algorithm or class structure, while novices would tend to memorize syntactic elements. The results of this study indicate that this might be true. We have found that the lookup count difference between novices and experts was significantly different but only for the items of the first group and for the item Class3. A possible explanation for this might be that with increasing length and complexity the experts were not able to recognize the underlying class structure or algorithm. Therefore, they might have needed to memorize the source code line by line as novices needed to do.

We also found that experts needed fewer lookups for the first class item but more lookups for the first algorithm than for the first normal text. An implication of this is the possibility that experts remember classes easier than algorithms. This evidence is also supported by the U test analysis of the lookup time difference. We have found that the lookup time difference between novices and experts was significantly different for the same items. It can thus be suggested that expert programmers are more familiar with the syntax of the programming language and therefore do not need to remember all syntactic elements which leads to less lookup time. It is also notable that expert programmers looked at the items of the first less than they looked at the normal text. Whereas the class items of the third group was looked at more than the normal text. This finding also suggests that experts memorize shorter source codes on a semantic level, therefore they needed less lookup time. However, when the source codes got increasingly longer and more complex expert programmers might only benefit from their familiarity in the syntax of the programming language. Therefore, it makes sense that the longest and most complex algorithm showed no difference between experts and novices. Hence our results are in line with Adelson's [8] but a

note of caution is due here since our experts had a significantly better working memory capacity than our novices. This could have influenced our results.

6 Conclusion

This study has identified that expert programmers are better in memorizing source code than novices. We also have found that expert programmers performed differently when presented long and complex source code than short and less complex source codes. These results support the idea that expert programmers memorize source code in semantic entities, whereas novice programmers memorize source code in syntactical entities as found by Adelson [8]. One source of weakness in this study which could have affected our results was the significant difference in working memory capacity between novices and experts. This issue of the correlation between working memory capacity and programming experience is an intriguing one which could be usefully explored in further research. There is, therefore, a definite need for conducting a similar study as presented in this paper, when the correlation between memory capacity and programming experience is clearer.

References

1. Klieme, E., Hartig, J., Rauch, D.: The concept of competence in educational contexts. Assessment of competencies in educational contexts. 3-22 (2008)
2. Koeppen, K., Hartig, J., Klieme, E., Leutner, D.: Current issues in competence modeling and assessment. Zeitschrift für Psychologie/Journal of Psychology. 216(2), 61-73 (2008)
3. Martens, K., Niemann, D.: When do numbers count? The differential impact of the PISA rating and ranking on education policy in Germany and the US. German Politics. 22(3), 314-332 (2013)
4. Weinert, F.E.: Concept of competence: A conceptual clarification. In Rychen, D., Salganik, S., Hersh, L. (eds.) Defining and selecting key competencies. Hogrefe & Huber Publishers, Ashland, OH, US (2001)
5. Leutner, D., Fleischer, J., Grünkorn, J., Klieme, E. (eds.): Competence Assessment in Education – Research, Models and Instruments. Springer International Publishing (2017)
6. Kramer, M.; Hubwieser, P.; Brinda, T.: A Competency Structure Model of Object-Oriented Programming. In: 2016 International Conference on Learning and Teaching in Computing and Engineering (LaTICE). IEEE, 1-8 (2016).
7. Kramer, M.; Tobinski, D.; Brinda, T.: On the Way to a Test Instrument for Object-Oriented Programming Competencies. In: Proceedings of the 16th Koli Calling International Conference on Computing Education Research. ACM, 145-149 (2016).
8. Adelson, B.: Problem solving and the development of abstract categories in programming languages. Memory & Cognition. 9(4), 422-433 (1981)
9. Kellog, R.T.: A model of working memory in writing. The cognitive demands of writing: processing capacity and working memory in text production. Amsterdam University Press. 57-71 (1996)
10. Hayes, J.R.: A new framework for understanding cognition and affect in writing. Perspectives on writing: Research, theory, and practice. 6 (1996)
11. Baddeley, A.D., Hitch, G.: Working memory. The Psychology of Learning and Motivation. 8, 47-89 (1974)
12. de Groot, A.: Thought and Choice in Chess. De Gruyter (2014)

13. Chase, W.G., Simon, H.A.: Perception in chess. Cognitive Psychology. 4(1), 55-81 (1973)
14. Corsi, P.: Human Memory and the Medial Temporal Region of the Brain. McGill University (1972)
15. Brunetti, R., Gatto, C.D., Delogu, F.: ecorsi: Implementation and testing of the corsi block-tapping task for digital tablets. Frontiers in Psychology. 5 (2014)
16. Beauducel, A.: Intelligence structure test: IST. Hogrefe (2009)
17. Bates, M.E., Lemay, E.P.: The d2 test of attention: construct validity and extensions in scoring techniques. Journal of the International Neuropsychological Society. 10(3), 392-400 (2004)
18. Levenshtein, V.: Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Soviet Physics Doklady. 10, 707 (1966)
19. Siegmund, J., Kästner, C., Liebig, J., Apel, S., Hanenberg, S.: Measuring and modeling programming experience. Empirical Software Engineering. 19(5), 1299-1334 (2014)
20. Misra, S., Adewumi, A.: Object-oriented cognitive complexity measures. In Handbook of Research on Innovations in Systems and Software Engineering. IGI Global, Hershey, PA (2015)
21. Amstad, T.: Wie verständlich sind unsere Zeitungen? University of Zurich (1978)
22. Flesch, R.: A new readability yardstick. Journal of Applied Psychology 32(3), 221-233 (1948)
23. Bamberger, R.: Lesen - verstehen - lernen - schreiben: die Schwierigkeitsstufen von Texten in deutscher Sprache. Jugend und Volk [u.a.]. Wien (1984)