Program Comprehension and Developers' Memory

Presentation of work originally published in the

Proc. of the 40th Intl. Conf. on Software Engineering

Jacob Krüger[1,2], Jens Wiemann[2], Wolfram Fenske[2], Gunter Saake[2], Thomas Leich[1,3]

Abstract: In this extended abstract, we summarize our paper "Do You Remember This Source Code?", published at the International Conference on Software Engineering 2018 [Kr18]. We discuss implications of our results on forgetting in the context of program comprehension, providing a more contextual perspective on our results compared to the original paper and a previous abstract [Kr19].

Keywords: Familiarity; forgetting; empirical study; maintenance; program comprehension

Software developers constantly design, implement, maintain, and re-engineer software systems. For this purpose, they have to understand the program itself, which is the most time-consuming and costly activity that software developers perform—called *program comprehension*. There exist numerous studies to show the impact of various factors on program comprehension, and techniques to facilitate this activity, for instance, based on comments or identifier names. However, few studies investigate the remaining *software familiarity* of developers with a system. This familiarity comprises knowledge about the code, design, architecture, and application of the software that is *forgotten* over time. While some techniques (e.g., expertise identification) consider factors related to forgetting (i.e., time, authorship), we are not aware of empirical analyses that show the impact of these factors on developers' memory.

We conducted an empirical study on the self-assessed familiarity of 60 open-source developers with source code they worked on [Kr18]. More precisely, we investigated whether the well-known forgetting model of Ebbinghaus applies to software familiarity, analyzed three factors (i.e., number of edits, ratio of own code, tracking behavior) from learning theory that should also influence forgetting, and computed the memory strength of our participants. In Table 1, we show the results of testing the significance of our observations. To this end, we used two rank correlations (Kendall's Tau is more strict than Spearman's Rho) that indicate the effect each factor has on our participants' memory. We remark that we controlled for the file sizes to ensure that these had no impact, as we asked our participants to state their remaining familiarity as ratios of the files.

[1] Harz University of Applied Sciences Wernigerode; tleich@hs-harz.de
[2] Otto-von-Guericke-University Magdeburg; jkrueger@ovgu.de; wfenske@ovgu.de; saake@ovgu.de
[3] METOP GmbH Magdeburg

Tab. 1: Spearman's Rho (r_s), Kendall's Tau (τ), and the corresponding significance (sig.) values.

Factor	r_s	sig.	τ	sig.
File Size	0.16	0.22	0.11	0.24
Number of Edits	0.67	4.56×10^{-9}	0.55	5.18×10^{-8}
Ratio of Own Code	0.55	4.57×10^{-6}	0.42	6.86×10^{-6}
Tracking Behavior	0.04	0.79	0.02	0.8

As we can see in Table 1, the file sizes did not correlate with familiarity, which was the prerequisite for our analysis. The remaining results showed a local maximum in familiarity after approximately 120 days had passed since the last edit of a file. This highlights the impact of factors other than time on familiarity. Our results indicate that the *number of edits*, which represents repeated commits to the same file at different days, has a moderate to strong, positive correlation to familiarity. For the *ratio of own code* that a developer implemented, we also found a moderate, positive correlation. Consequently, we assume that repeatedly working on a file and implementing more of its code does indeed improve a developer's memory of that file. In contrast, we found no correlation for the *tracking behavior*, which refers to developers following and analyzing the changes others employ on the file. However, this may be due to different understandings of tracking and we need to conduct further analyses to see whether this finding holds true. Finally, we found that the averaged curve resembles that of Ebbinghaus, especially if we remove the number of edits as factor. Thus, Ebbinghaus' forgetting model seems to apply to software engineering, but only if developers implemented a file in one session. Overall, we argue that we need an adapted forgetting curve for software development that considers additional factors that are specific to the activities, processes, and interactions of developers.

We conducted this research in the context of re-engineering, for which developers first need to comprehend the software. As program comprehension is costly, the costs for re-engineering heavily depend on the remaining familiarity a developer has. In practice, we can find various scenarios in which the question arises who should perform a task. For example, consider a developer who implemented a file some time ago. Other developers made changes to this file, introducing new functionality or fixing bugs. The question is, who is best suited to perform a new task on this file? It may be the original developer (ratio of own code), the one who did most changes (number of edits) or the one with the most recent changes (time). This question is impossible to answer precisely, but our research indicates that existing heuristics will benefit from taking the impact of forgetting on software familiarity and program comprehension into account.

Acknowledgments: Supported by DFG grants LE 3382/2-1 and SA 465/49-1.

References

[Kr18] Krüger, Jacob; Wiemann, Jens; Fenske, Wolfram; Saake, Gunter; Leich, Thomas: Do You Remember This Source Code? In: ICSE. 2018.

[Kr19] Krüger, Jacob; Wiemann, Jens; Fenske, Wolfram; Saake, Gunter; Leich, Thomas: Understanding How Programmers Forget. In: SE/SWM. 2019.