A Neuro-Cognitive Perspective of Program Comprehension

Norman Peitek
Leibniz Institute for Neurobiology, Magdeburg, Germany

ABSTRACT

Program comprehension is the cognitive process of understanding code. Researchers have proposed several models to describe program comprehension. However, because program comprehension is an internal process and difficult to measure, the accuracy of the existing models are limited. Neuro-imaging methods, such as functional magnetic resonance imaging (fMRI), provide a novel neuro-cognitive perspective to program-comprehension research. With my thesis work, we aim at establishing fMRI as a new tool for program-comprehension and software-engineering studies. Furthermore, we seek to refine our existing framework for conducting fMRI studies by extending it with eye tracking and improved control conditions. We describe how we will apply our upgraded framework to extend our understanding of program comprehension. In the long-run, we would like to contribute insights from our fMRI studies into software-engineering practices by providing code-styling guidelines and programming tools, which reduce the required cognitive effort to comprehend code.

CCS CONCEPTS

• **Human-centered computing → HCI design and evaluation methods**; **Empirical studies in HCI**;

KEYWORDS

program comprehension, top-down comprehension, functional magnetic resonance imaging, eye tracking

ACM Reference Format:
Norman Peitek. 2018. A Neuro-Cognitive Perspective of Program Comprehension. In *ICSE '18 Companion: 40th International Conference on Software Engineering, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183440.3183442

1 INTRODUCTION

Program comprehension is an important cognitive process, because programmers spend a lot of their time with comprehending source code [12]. It is also an extensively studied process: in the last few decades, hundreds of papers have been published developing and refining program-comprehension models [3, 6, 20, 27, 28].

Understanding how programmers comprehend code is inherently difficult because internal processes cannot be measured directly. Thus, traditional research measures, which have been used to develop program-comprehension models (e.g., top-down comprehension or bottom-up comprehension), cannot capture all facets of program comprehension. Hence, in recent years, researchers have been exploring how physiological measures can add a new perspective to program-comprehension research.

We contribute to this trend by developing guidelines to establish functional magnetic resonance imaging (fMRI) as standard instrument to observe program comprehension and other related processes. fMRI is an established measurement technique in neuroscience. It allows researchers to observe brain activation in terms of activated brain areas and activation strength [11]. Based on a participant's brain activation, we can infer corresponding cognitive processes [9]. In our research, we use fMRI to unravel the underlying cognitive processes of program comprehension, helping to unify existing models of program comprehension and striving for a deeper understanding of how developers understand code. In the long run, an understanding of how programmers comprehend code allows us to boost programmers' productivity with better designed tools, which reduce the required cognitive effort.

To highlight the benefits of fMRI in program-comprehension research, we introduce important program-comprehension models in Section 2, in which we also present an overview of other recently used physiological measures in program-comprehension research. Next, we delineate the goals and contributions of this thesis research in Section 3. In Section 4, we describe current results and projects, and outline planned studies in the future. We conclude the paper in Section 5.

2 PROGRAM COMPREHENSION

Program-comprehension models can be categorized into different categories [6]. In this section, we describe the two most important categories for our research, bottom-up and top-down-comprehension models.

2.1 Bottom-Up Comprehension

Bottom-up comprehension describes the most basic strategy of understanding software [20]: programmers read code line by line, understand and extract the meaning of each statement, and then group the semantic information of each individual statement to a larger entity (i.e., *chunking* [23]). This recursive process is repeated until programmers achieve a complete understanding of a code [20].

Since a programmer has to read every single statement of a code, individually comprehend it, and then assimilate all statements until a final, abstract understanding is achieved, this process is slow and tedious. In other words, bottom-up comprehension requires a high cognitive effort of programmers [26], so they typically try to employ top-down comprehension, which we explain next.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05…$15.00
https://doi.org/10.1145/3183440.3183442

2.2 Top-Down Comprehension

Unlike bottom-up comprehension, top-down comprehension is a fast and efficient process [2, 3, 27]. Programmers use prior experience and domain knowledge to quickly gain an understanding of a code's intent. Programmers look for *beacons* (i.e., "sets of features that typically indicate the occurrence of certain structures or operations in the code" [3]) to build an initial hypothesis of a code's intent. Then, programmers validate and refine their hypothesis by quickly jumping between the code's points of interest (e.g., input variables, loop structures, return variable). If the implementation confirms their hypothesis, programmers have quickly and efficiently understood a code's intent. If an unexpected implementation contradicts their initial hypothesis, programmers have to fall back to bottom-up comprehension.

Top-down comprehension requires lower cognitive effort than bottom-up comprehension, since the process starts with an initial hypothesis and statements do not need to be understood separately [26]. Because of missing experience novices often cannot employ top-down comprehension and tend to use bottom-up comprehension [27].

2.3 Limitations and New Physiological Measures

The models developed in the past are based on observational data, such as think-aloud protocols [20] or objective performance measures [27]. However, the insights from these research measures cannot capture the entirety of the complex cognitive processes involved in program comprehension. For example, cognitive effort is subjective and difficult to assess with traditional measures. Thus, we and others explore new physiological measures to better grasp the cognitive aspect of program comprehension.

One of the first measures was eye tracking, which provides insight into the programmer's visuo-spatial attention. For example, Sharif and Maletic used eye tracking to study the effect of camelCase and under_score identifiers on efficiency of program comprehension [22]. Electroencephalography (EEG) is a method for observing the cognitive effort by measuring the electrical activity of the brain. For example, Crk and others used EEG to examine the difference of cognitive effort based on programming experience [4]. Functional near-infrared spectroscopy (fNIRS) observes brain activity based on the same underlying principle as fMRI, but with a lower spatial resolution. Nakagawa and others used fNIRS to measure cerebral blood flow during program comprehension [15]. Finally, the measures have been combined: Lee and others used EEG and eye tracking to predict programmer expertise, which provided empirical evidence for the validity of hybrid comprehension models [13].

Siegmund and others were the first to use fMRI when they developed a framework for studying program comprehension with fMRI and applied it to provide a neuro-cognitive perspective to bottom-up comprehension [24, 25]. This inspired Floyd and others to study program comprehension and contrast it to reviewing code and prose [8]. Duraes and others also used fMRI to study the awareness of programmers regarding errors during debugging [5]. The four fMRI studies identified largely overlapping networks of activated brain areas and associated cognitive processes [5, 8, 25, 26].

The results of these early fMRI studies are promising and encourage us to deepen and extend the neuro-cognitive perspective on program comprehension.

3 ADVANCING RESEARCH ON PROGRAM COMPREHENSION USING FMRI

We strive to continue the initial research of using fMRI in program-comprehension studies to establish it as a tool for software-engineering research. We would like to advance it in two directions:

First, we will refine Siegmund's fMRI study framework. Only with a mature framework we can tackle more advanced research questions (e.g., influence of beacons) and motive other researchers in software engineering to consider fMRI as a measure.

Second, top-down comprehension and its efficiency is particularly interesting to understand programmer productivity and support novices in learning to program. We will apply the fMRI study framework to understand when top-down comprehension is possible and how it is affected by code aspects and a programmer's experience. This line of research allows us to refine and extend existing top-down-comprehension models, and in the long run, support programmers' productivity.

My thesis research makes the following contributions:

- An improved methodological framework for future fMRI studies of program comprehension, which allows us to also study fine-grained effects (e.g., influence of beacons). Furthermore, we will share replication packages and insights for each study to pave the way for other researchers to also conduct fMRI studies of program comprehension.
- Refinement of top-down-comprehension models, in particular the impact of code aspects (e.g., beacons, layout, indentations) and the influence of programmer experience.
- A recommendation of code styling and programmer tools based on our neuro-cognitive perspective of program comprehension.

In the next section, we outline the plan for achieving these contributions.

4 RESEARCH TIMELINE AND METHODS

The research described in this paper is already under way. Thus, we split this section into two parts: First, we discuss already published results and current studies in Section 4.1. Second, we outline the upcoming studies and their research questions in Section 4.2.

4.1 Current Results and Studies

4.1.1 Neural Efficiency of Top-Down Comprehension. In our recent fMRI study, we observed a difference in activation between bottom-up and top-down comprehension [26]. The key result is the higher neural efficiency[1] of top-down program comprehension: We found that the same brain areas are activated independent of top-down or bottom-up comprehension, but the activation strength was significantly lower for top-down comprehension. This provides empirical evidence on the reduced cognitive effort of programmers

[1]Neural efficiency is described as "brighter individuals display lower (more efficient) brain activation while performing cognitive tasks", but can also be an effect of learning [16].

during top-down comprehension. We also strengthened the validity of the initial fMRI study framework by successfully replicating the results of Siegmund's first fMRI study [25]. Our third research goal was to understand the effect of code styling and beacons on top-down comprehension.

However, we could not find the expected influence of code styling and beacons. Based on the data and comments of participants, we concluded that the current study framework is probably too limited to capture such small effects. We discuss next how we plan to overcome this limitation.

4.1.2 Enhancing fMRI Studies. To study fine-grained effects of top-down comprehension (e.g., influence of beacons or code styling), the fMRI study framework needs to be enhanced to capture subtle differences. We currently evaluate an extension of the framework by:

- Eye tracking, so that we can, for example, observe how programmers look for beacons. Eye tracking can also help us to explain why brain activation changes throughout a program-comprehension task [19], for example, when it shows that a programmer is fixating on a beacon.
- Suppression of activation during the rest condition. The rest condition is necessary to let a participant's brain activation return to its baseline. However, the naturally occurring reflection after a task and strategy planning of our study participants during the rest condition delays the return to the baseline and thus reduces the contrast strength. A d2 task[2] may suppress this counterproductive behavior during the rest condition. Integrating a d2 task after each program-comprehension task would increase the contrast strength during fMRI data analysis and make subtle differences in cognitive processes more detectable.
- Finding a suitable control condition. A control condition is necessary to filter out brain activation that is unspecific for program comprehension. For example, looking at code will cause an activation in the visual cortex, but does not reveal cognitive processes, which are interesting for program-comprehension research. Past fMRI studies of program comprehension have been using locating syntax errors as control condition for the visual input [25, 26]. However, it is possible that programmers still (partially) comprehend code when finding syntax errors. This is problematic, because it reduces the fMRI contrast strength and diminishes our chances of observing fine-grained effects.

4.1.3 Influence of Code on Top-Down Comprehension. Our fMRI study showed a difference between top-down and bottom-up comprehension, but failed to observe smaller effects [26]. We can tackle more fine-grained questions with our improved framework. For example, we are currently planning to replicate this study with the improved framework to understand the effect of beacons on top-down comprehension.

Furthermore, eye-tracking data offer a further perspective: It can also help us to identify effects on brain activation. For example, do we see an increase in a memory-related brain areas when participants fixate on a beacon? Combining eye tracking and fMRI has the potential to explain *why* there is a difference in brain activation at a specific point in time. In particular, we can understand changing strength in brain activation observed throughout a program-comprehension task. Eventually, we aim at identifying the phases of program comprehension and accordingly extend and refine existing models.

In this study, we will also go beyond the influence of comprehension strategy (i.e., bottom-up vs. top-down comprehension) and activated brain areas: We seek to understand how a programmer's experience, measured with a validated questionnaire [7], influences the *strength* of brain activation and deactivation. We expect that novices need to concentrate more than experts, and thus display a stronger deactivation in brain areas of the *default mode network*[3]. Research with EEG showed differences in cognitive effort between novice and expert programmers [4]. Many years of training do not only lead to a higher neural efficiency, but can also affect brain structure [17]. If we can replicate the differences in cognitive effort and brain structures for programmers with fMRI, we would be able to estimate the effect of a programmer's experience as a confounding variable. Furthermore, objectively evaluating a programmer's experience based on their brain activation and structure may offer future applications beyond research (e.g., teaching).

4.2 Future Work

Besides the current results and two ongoing studies, we plan to explore further aspects of top-down comprehension in the near future and, eventually, build on our results to develop approaches to support programmers' productivity.

4.2.1 Role of Memory during Top-Down Comprehension. Program-comprehension theories hypothesize that programmers are required to have programming *plans* (i.e., "program fragments that represent stereotypic action sequences in programming" [27]) in their minds to permit top-down comprehension. With fMRI, we should observe memory-related brain activation for experienced developers during top-down comprehension. In this line of work, we seek to understand the role of memory during top-down comprehension. What level of experience and domain knowledge is necessary for programmers to ingrain abstract models in their brain? An answer to this question allows us to better understand when top-down comprehension is possible and what kind of programmers are able to employ top-down comprehension.

4.2.2 Influence of Experience on Top-Down Comprehension. A programmer's experience itself is an interesting factor. In our previous studies, the influence of experience was a secondary objective due to a rather homogeneous participant group of computer science students. To take a closer look at the role of programming experience, we plan to observe novice and experienced developers. This way, we can explore whether experienced developers show a higher neural efficiency during program comprehension. Additionally, we can explore differences in novice and expert top-down program comprehension [18]. What is the difference between novice and

[2] d2 is a psychological test of attention in which participants scan through a row of letters and decide for each letter whether it is a *d* with two marks [1].

[3] The brain areas of the default mode network show strong activation during rest conditions. When a participant focuses on a cognitively difficult task, the activation in the default mode network is reduced and data will show a *deactivation* [14].

experienced programmers when they employ top-down comprehension in terms of activated brain areas and activation strength? Are the phases of program comprehension, which we identified in a previous study, different for the two groups? This line of research will help us to identify how different novices and experienced developers understand code.

4.2.3 Toward Code Styling and Tool Support. Our insight from previous studies, in particular regarding the influence of code styling, memory, and experience, potentially allows us to contribute to software-engineering practices. For example, we could explore code-styling guidelines for minimizing cognitive effort needed for program comprehension.

Furthermore, using our previous results, we would like to support developers' cognitive processes by presenting code in a state-of-the-art way (e.g., *semantic* highlighting of beacons). Extending the dry code display with algorithm visualizations may allow programmers to offload cognitive effort to other brain areas [10, 21]. This leads to the overarching research question whether we can guide inexperienced developers toward efficient top-down comprehension. If so, this would have implications for enhancing integrated development environments (IDEs).

The capstone of this thesis research is a prototype of a custom IDE plugin. *CognitiveIDE* will be the result of all insights from our studies on top-down comprehension. It uses unique ways of presenting the code (e.g., semantic highlighting, algorithm visualization) and consequently allows programmers to stay longer in top-down comprehension.

5 CONCLUSION

Program comprehension, especially top-down comprehension, is an important cognitive process, because it is what programmers do most of their time. In our work, we add another perspective to program-comprehension research by applying fMRI and developing guidelines to establish it as standard research method. Eventually, this will help us to understand how programmers think when working with code.

This line of research has a long way to go, but the next steps are well-defined. Ultimately, it helps us to increase productivity with code styling and programmer tools that are tailored to programmers' way of thinking processes.

ACKNOWLEDGMENTS

I would like to thank Janet Siegmund, André Brechmann, Sven Apel, and Chris Parnin for the fruitful discussions and helpful feedback on this paper. The author's work is supported by DFG grants SI 2045/2-1 and BR 2267/7-1.

REFERENCES

[1] Rolf Brickenkamp, Lothar Schmidt-Atzert, and Detlev Liepmann. 2010. *Test d2-Revision: Aufmerksamkeits-und Konzentrationstest.* Hogrefe Göttingen.
[2] Ruven Brooks. 1978. Using a Behavioral Theory of Program Comprehension in Software Engineering. In *Proc. Int'l Conf. Software Engineering (ICSE).* IEEE CS, 196–201.
[3] Ruven Brooks. 1983. Towards a Theory of the Comprehension of Computer Programs. *Int'l Journal of Man-Machine Studies* 18, 6 (1983), 543–554.
[4] Igor Crk, Timothy Kluthe, and Andreas Stefik. 2015. Understanding Programming Expertise: An Empirical Study of Phasic Brain Wave Changes. *ACM Trans. Comput.-Hum. Interact.* 23, 1, Article 2 (Dec. 2015), 29 pages.
[5] J. Duraes, H. Madeira, J. Castelhano, C. Duarte, and M. C. Branco. 2016. WAP: Understanding the Brain at Software Debugging. In *Proc. Int'l Symposium Software Reliability Engineering (ISSRE).* IEEE CS, 87–92.
[6] Christopher Exton. 2002. Constructivism and Program Comprehension Strategies. In *Proc. Int'l Workshop Program Comprehension (IWPC).* IEEE Computer Society, Washington, DC, USA, 281–284.
[7] Janet Feigenspan, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. 2012. Measuring Programming Experience. In *Proc. Int'l Conf. Program Comprehension (ICPC).* IEEE CS, 73–82.
[8] Benjamin Floyd, Tyler Santander, and Westley Weimer. 2017. Decoding the Representation of Code in the Brain: An fMRI Study of Code Review and Expertise. In *Proc. Int'l Conf. Software Engineering (ICSE).* IEEE Press, Piscataway, NJ, USA, 175–186.
[9] Michael S. Gazzaniga, Richard B. Ivry, and George R. Mangun. 2013. *Cognitive Neuroscience: The Biology of the Mind.* Norton & Company.
[10] Dean Hendrix, JH Cross, and Saeed Maghsoodloo. 2002. The effectiveness of control structure diagrams in source code comprehension activities. *IEEE Transactions on Software Engineering* 28, 5 (2002), 463–477.
[11] Scott Huettel, Allen Song, and Gregory McCarthy. 2014. *Functional Magnetic Resonance Imaging.* Vol. 3. Sinauer Associates.
[12] Thomas D. LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining Mental Models: A Study of Developer Work Habits. In *Proc. Int'l Conf. Software Engineering (ICSE).* ACM, New York, NY, USA, 492–501.
[13] Seolhwa Lee, Danial Hooshyar, Hyesung Ji, Kichun Nam, and Heuiseok Lim. 2017. Mining Biometric Data to Predict Programmer Expertise and Task Difficulty. *Cluster Computing* (2017), 1–11.
[14] K. McKiernan, J. Kaufman, J. Kucera-Thompson, and J. Binder. 2003. A Parametric Manipulation of factors Affecting Task-Induced Deactivation in Functional Neuroimaging. *J. Cognitive Neuroscience* 15, 3 (2003), 394–408.
[15] Takao Nakagawa, Yasutaka Kamei, Hidetake Uwano, Akito Monden, Kenichi Matsumoto, and Daniel M. German. 2014. Quantifying Programmers' Mental Workload During Program Comprehension Based on Cerebral Blood Flow Measurement: A Controlled Experiment. In *Proc. Int'l Conf. Software Engineering (ICSE).* ACM, 448–451.
[16] Aljoscha C. Neubauer and Andreas Fink. 2009. Intelligence and neural efficiency. *Neuroscience & Biobehavioral Reviews* 33, 7 (2009), 1004 – 1023. https://doi.org/10.1016/j.neubiorev.2009.04.001
[17] Nicola Neumann, Martin Lotze, and Simon B Eickhoff. 2016. Cognitive Expertise: An ALE Meta-Analysis. *Human brain mapping* 37, 1 (2016), 262–272.
[18] Chris Parnin, Janet Siegmund, and Norman Peitek. 2017. On the Nature of Programmer Expertise. PPIG.
[19] Norman Peitek, Janet Siegmund, and André Brechmann. 2017. Enhancing fMRI Studies of Program Comprehension with Eye-Tracking. In *Proc. Int'l Workshop on Eye Movements in Programming.* Freie Universität Berlin, 22–23.
[20] Nancy Pennington. 1987. Stimulus Structures and Mental Representations in Expert Comprehension of Computer Programs. *Cognitive Psychology* 19, 3 (1987), 295–341.
[21] Yuri Sato, Sayako Masuda, Yoshiaki Someya, Takeo Tsujii, and Shigeru Watanabe. 2015. An fMRI Analysis of the Efficacy of Euler Diagrams in Logical Reasoning. In *Symp. on Visual Languages and Human-Centric Computing (VL/HCC).* IEEE CS.
[22] Bonita Sharif and Johnathon Maletic. 2010. An Eye Tracking Study on camelCase and under_score Identifier Styles. In *Proc. Int'l Conf. Program Comprehension (ICPC).* IEEE CS, 196–205.
[23] Ben Shneiderman and Richard Mayer. 1979. Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results. *Int'l J. Parallel Programming* 8, 3 (1979), 219–238.
[24] Janet Siegmund, André Brechmann, Sven Apel, Christian Kästner, Jörg Liebig, Thomas Leich, and Gunter Saake. 2012. Toward Measuring Program Comprehension with Functional Magnetic Resonance Imaging. In *Proc. Int'l Symposium Foundations of Software Engineering–New Ideas Track (FSE-NIER).* ACM, 24:1–24:4.
[25] Janet Siegmund, Christian Kästner, Sven Apel, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, and André Brechmann. 2014. Understanding Understanding Source Code with Functional Magnetic Resonance Imaging. In *Proc. Int'l Conf. Software Engineering (ICSE).* ACM, 378–389.
[26] Janet Siegmund, Norman Peitek, Chris Parnin, Sven Apel, Johannes Hofmeister, Christian Kästner, Andrew Begel, Anja Bethmann, and André Brechmann. 2017. Measuring Neural Efficiency of Program Comprehension. In *Proc. of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017).* ACM, New York, NY, USA, 140–150.
[27] Elliot Soloway and Kate Ehrlich. 1984. Empirical Studies of Programming Knowledge. *IEEE Trans. Softw. Eng.* 10, 5 (1984), 595–609.
[28] Anneliese von Mayrhauser and Marie Vans. 1995. Program Comprehension During Software Maintenance and Evolution. *Computer* 28, 8 (1995), 44–55.