Report from Dagstuhl Seminar 22402

Foundations for a New Perspective of Understanding Programming

Madeline Endres[*1], André Brechmann[†2], Bonita Sharif[†3], Westley Weimer[†4], and Janet Siegmund[†5]

1 **University of Michigan – Ann Arbor, US.** endremad@umich.edu
2 **Leibniz-Institut fur Neurobiologie – Magdeburg, DE.** brechmann@lin-magdeburg.de
3 **University of Nebraska – Lincoln, US.** bsharif@unl.edu
4 **University of Michigan – Ann Arbor, US.** weimerw@umich.edu
5 **TU Chemnitz, DE.** janet.siegmund@informatik.tu-chemnitz.de

Abstract

Software is created by people who think, feel, and express themselves to one another and their computers. For a long time, researchers have investigated how people read and write code on their computers and talk about code with one another. This way, researchers identified skills, education, and practices necessary to acquire expertise and perform software development duties. While these investigations are valuable, we have yet to devise and validate a scientific theory of *program comprehension*, which would be an important step in designing support for developers that is tailored to their cognitive needs. To succeed, we need techniques to shed more light on how programmers think. To this end, we need to look beyond computer science research.

Specifically, in the field of psychology and cognitive neuroscience, considerable progress has been made in building theories of cognitive processes. Important enabling technologies include eye tracking, functional magnetic resonance imaging (fMRI), electroencephalography (EEG), and functional near infrared spectroscopy (fNIRS). These methods have revolutionized the understanding of cognitive processes and are routinely used in non-computing disciplines. Such techniques have the potential to also modernize classic approaches to program comprehension research by informing new experimental designs. However, the use of such technologies to study program comprehension is recent, and many of the challenges of this interdisciplinary field remain unexplored.

This report documents the program and the outcomes of Dagstuhl Seminar 22402, "Foundations for a New Perspective of Understanding Programming", which explores these challenges. In total, 23 on-site participants attended the seminar along with two virtual keynote speakers. Participants engaged in intensive collaboration, including discussing past and current research, identifying gaps in the literature, and proposing future directions for improving the state of the art in program comprehension research.

Seminar October 3–7, 2022 – http://www.dagstuhl.de/22402

2012 ACM Subject Classification General and reference → General literature; General and reference → Empirical studies; Software and its engineering → Software design engineering; Human-centered computing → User studies

Keywords and phrases Programming Methodology, Programming Education, Program Comprehension, Neuro-imaging, Eye Tracking, Human Cognition, Human Computer Interaction, Software Engineering, Human Factors

Digital Object Identifier 10.4230/DagRep.12.10.61

[*] Editorial Assistant / Collector
[†] Editor / Organizer

Foundations for a New Perspective of Understanding Programming, *Dagstuhl Reports*, Vol. 12, Issue 10, pp. 61–83
Editors: Madeline Endres, André Brechmann, Bonita Sharif, Westley Weimer, and Janet Siegmund

Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Executive Summary

Madeline Endres (University of Michigan – Ann Arbor, US)
André Brechmann (Leibniz-Institut fur Neurobiologie – Magdeburg, DE)
Bonita Sharif (University of Nebraska – Lincoln, US)
Westley Weimer (University of Michigan – Ann Arbor, US)
Janet Siegmund (TU Chemnitz, DE)

The goal of the seminar *Foundations for a New Perspective of Understanding Programming* was to address how to modernize the perspective on program comprehension and thus make progress regarding our understanding of it. We focused on two challenges: First, we discussed how to provide guidance on addressing methodological issues in interdisciplinary program comprehension research. Second, we aimed at defining a unifying enumeration of the dimensions of a neuroscientific perspective on program comprehension, such that researchers are able to more systematically investigate gaps in the literature.

Through individual participant presentations and the resulting group discussions, we identified several relevant aspects that we discussed further in dedicated working groups. These included discussing how to make better use of eye tracking (Section 4.1), identifying the role of readability for program comprehension (Section 4.2), and considering how machine learning could help to develop a model of program comprehension (Section 4.3). These working groups helped us address the first goal of the seminar by providing guidance on program comprehension research methodologies and identifying potential next steps. To conclude the seminar, participants discussed a possible taxonomy for program comprehension research (Section 5.1). This taxonomy identifies commonalities and differences across a broad research area, and addresses our second goal of helping to unify the program comprehension research space; it can serve as a starting point for future researchers to build on to develop an understanding of program comprehension. Also, for researchers entering this field, it is a first glimpse of the complexity of understanding and researching program comprehension.

Beyond identifying research problems in program comprehension research, the many collaborative sessions at this seminar generated numerous potential multi-institutional and interdisciplinary collaborations. We hope that, by making progress on the program comprehension research challenges, we help bring this new research direction one step closer to becoming standard in programming research and disseminating it to a wider audience.

2 Table of Contents

3 Overview of Talks

3.1 Brains on Code: A Neuroscientific Foundation on Program Comprehension

Sven Apel (Universität des Saarlandes – Saarbrücken, DE)

Research on program comprehension has a fundamental limitation: program comprehension is a cognitive process that cannot be directly observed, which leaves considerable room for misinterpretation, uncertainty, and confounders. In Brains On Code, we are developing a neuroscientific foundation of program comprehension. Instead of merely observing whether there is a difference regarding program comprehension (e.g., between two programming methods), we aim at precisely and reliably determining the key factors that cause the difference. This is especially challenging as humans are the subjects of study, and interpersonal variance and other confounding factors obfuscate the results.

The key idea of Brains On Code is to leverage established methods from cognitive neuroscience to obtain insights into the underlying processes and influential factors of program comprehension. Brains On Code pursues a multimodal approach that integrates different neuro-physiological measures as well as a cognitive computational modeling approach to establish the theoretical foundation. This way, Brains On Code lays the foundations of measuring and modeling program comprehension and offers substantial feedback for programming methodology, language design, and education. With Brains On Code, addressing longstanding foundational questions such as "How can we reliably measure program comprehension?", "What makes a program hard to understand?", and "What skills should programmers have?" comes into reach. Brains On Code does not only help answer these questions, but also provides an outline for applying the methodology beyond program code (models, specifications, requirements, etc.).

3.2 Shared Intentionality in Program Comprehension

Andrew Begel (Carnegie Mellon University – Pittsburgh, US)

Observing communication is a revealing way to indicate comprehension about code at many different abstraction levels. Using an analytic lens from linguistics, we can precisely describe this communication and thus enable us to make inferences about a person's comprehension of a program. Not only are the speaker's actions important, but the agency of the listener is vital to establishing a desired states of shared attention (i.e., both parties are thinking about the same thing) and shared intentionality (i.e., the recursive knowledge that they both know the other is thinking about the same thing they are). When both speaker and listener communicate together, they can begin to comprehend code and take actions on it as a single distributed cognitive unit. The pair's joint knowledge can be used to execute changes to the code that may have been difficult or impossible for each of them apart.

The effectiveness of this kind of communication is not robust however, when one or both members of the pair identify with physical or cognitive disabilities, e.g., a programmer is blind or low vision, or another has ADHD or dyslexia. In our research, we employ AI techniques in

computer vision, speech recognition, NLP, and physiological sensors to interpret, translate, and convey information between speaker and listener. This increases the likelihood that people of mixed abilities can successfully communicate about code, achieving a desired state of shared intentionality that illustrates their joint distributed comprehension and enables them to efficiently make changes together.

3.3 Insights into Program Comprehension with EEG and Eye tracking

Annabelle Bergum (Universität des Saarlandes – Saarbrücken, DE)

Main reference Norman Peitek, Annabelle Bergum, Maurice Rekrut, Jonas Mucke, Matthias Nadig, Chris Parnin, Janet Siegmund, Sven Apel: "Correlates of programmer efficacy and their link to experience: a combined EEG and eye-tracking study", in Proc. of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022, pp. 120–131, ACM, 2022.
URL https://doi.org/10.1145/3540250.3549084

One of the main research questions in our research group is "How can we reliably measure program comprehension?". To answer this question, fMRI and eye tracking studies were conducted. When I joined the team, in 2021, we enlarged the research field from our research group from fMRI to EEG. Since then, we conducted two EEG studies. The first study included the challenge of the new methodology which comes along with using EEG instead of fMRI. In the second study, we took a closer look at the influence of the baseline. We thereby adapted our study design to incorporate four different baselines. Therefore, we can afterwards investigate the effect of the baseline within one study, eliminating a lot of other influencing factors.

3.4 Linking fMRI research on sequential processing and category learning to understanding programming

André Brechmann (Leibniz-Institut für Neurobiologie – Magdeburg, DE)

When starting to work on understanding program comprehension using fMRI ten years ago in collaboration with Janet Siegmund, there was no blueprint how to approach the topic experimentally. First, I had to learn that empirical research was not very abundant at that time even though programming is such an important economic and societal topic. In the previous years, however, we have seen much progress in empirical research on programming, including brain imaging. Now it is about time for in depth discussions on how to pursue the topic further by teaming up with interested researchers from different disciplines and to start developing a theory of programming. In my talk (see attached slides) I contribute my neuroscientific perspective on program comprehension and discuss my view of sequential processing and cognitive sequencing as key component of programming and how to deal with the dynamics and individuality of program comprehension based on my experience from studying the dynamics of rule-based category learning.

3.5 Studying Eye Movements During Code Reading

Teresa Busjahn (Hochschule für Technik und Wirtschaft Berlin – Berlin, DE)

Main reference Teresa Busjahn: "Empirical analysis of eye movements during code reading: evaluation and development of methods", 2021.
URL https://doi.org/10.17619/UNIPB/1-1118

Studying eye movements during reading provides valuable insights into natural-language text comprehension and also lends itself to application in program comprehension. Using English and Java as exemplary languages, differences can be found between natural-language text and code reading, as well as in how novice and expert programmers read code. For instance, when reading natural-language text, a larger part of the text is looked at directly than when reading code. Moreover, during code reading, expert programmers attend to the main-method much sooner than novices. However, this line of research also brings about methodological challenges like event detection and correction of spatial errors. Addressing these is an ongoing effort.

3.6 Using Physiological Measures to Identify Cognitive States

Martha E. Crosby (University of Hawaii at Manoa – Honolulu, US) and Jan Stelovsky (University of Hawaii at Manoa – Honolulu, US)

The way that humans interact and absorb information delivered through technology is of interest to researchers in many fields. The accurate assessment of cognitive states such as arousal, fatigue, stress, task difficulty is essential to identifying and defining cognitive processes and testing models of how cognitive processes operate and interact. The individual and the situation can affect the measurement of cognitive states from a single type of sensor. Measurements from multiple sensors, when combined, produce a more robust measurement of cognitive states (CS) such as mental overload. Adaptive filtering or other techniques of CS can then be used to potentially find misconceptions and potentially improve task performance. For the last several years, we have designed and executed experiments about individual differences of the way people perceive, search, and understand information presented in various multimedia environments. We have used a suite of passive physiological sensors (eye fixations, skin conductivity, body temperature, heart wave form, electroencephalography, and pressures on mouse) to better understand the processes that facilitate seeking, filtering, and shaping relevant information during tasks such as understanding computer programs. To understand and solve problems, programmers must have a level of program comprehension established. There are many variations and levels of program comprehension, dependent on individual programmers as well the specific code itself. Our research focuses on deriving changes in cognitive state information from the patterns of data acquired from the user from physiological sensors. If companies incorporate technology already available in Augmented Reality glasses and cell phones, CS information could potentially be used to give feedback in classroom or industry settings. For example, if many programmers show confusion or are misled by the code being reviewed, it may indicate there is a need to improve various aspects of it such as documentation or style.

3.7 Finding Flocus: Using Logs Data to Identify When Software Engineers Experience Flow or Focused Work

Sarah D'Angelo (Google – Mountain View, US)

Joint work of Sarah D'Angelo, Adam Brown, Ben Holtz, Ciera Jaspan, Collin Green

The concept of flow has been studied for decades across a wide variety of contexts from work to hobbies, and is a critical aspect of engineering productivity, however non-disruptively measuring flow has remained difficult. In this work, we take a mixed methods approach to understanding and measure how software engineers experience flow. We introduce a logs based metric called "flocus" that leverages machine learning and a comprehensive collection of logs data to identify periods of related actions (indicating focused behavior), and validate this metric against self-reported time in flow or focus using diary data and quarterly survey data. Our results indicate that we can determine when software engineers at a large technology company experience flow or focus using flocus. Extending this approach to incorporate other signals such as physiological data or eye tracking has the potential to get us closer to measuring a flow state.

3.8 Tracking Eye Movements in Programming

Andrew Duchowski (Clemson University, US)

Main reference Krzysztof Krejtz, Andrew T. Duchowski, Katarzyna Wisiecka, Izabela Krejtz: "Entropy of Eye Movements While Reading Code or Text", in Proc. of the 10th IEEE/ACM International Workshop on Eye Movements in Programming, EMIP@ICSE 2022, Pittsburgh, PA, USA, May 18-24, 2022, pp. 8–14, IEEE, 2022.
URL https://ieeexplore.ieee.org/document/9808970

The keynote, while focused on eye movements in programming, covers a wide variety topics, including: (1) basics of eye movements, (2) basic metrics, (3) advanced metrics, (4), cognitive load, (5) eye movements in programming, and (6) future challenges.

3.9 How Do New Programmers Understand Programs?

Madeline Endres (University of Michigan – Ann Arbor, US)

Joint work of Madeline Endres, Zachary Karas, Xiaosu Hu, Ioulia Kovelman, Madison Fransher, Priti Shah, Westley Weimer
Main reference Madeline Endres, Zachary Karas, Xiaosu Hu, Ioulia Kovelman, Westley Weimer: "Relating Reading, Visualization, and Coding for New Programmers: A Neuroimaging Study", in Proc. of the 43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021, pp. 600–612, IEEE, 2021.
URL https://doi.org/10.1109/ICSE43902.2021.00062

Understanding how novices reason about coding at a neurological level has implications for training the next generation of software engineers. I first briefly talk about our work using neuroimaging (fNIRS) to measure the neural activity associated with introductory programming. In this work, we relate brain activity when coding to that of reading natural

language or spatial visualization. In contrast to some studies with more expert programmers, we find that while programming, reading, and spatial visualization are all neurologically distinct for novices, there are more significant differences between prose and coding than between spatial visualization and coding. We also find a neural activation pattern predictive of programming performance 11 weeks later. I conclude with a discussion of future education-related directions I hope neuroimaging research explores going forward, including understanding how external factors such as native natural language, or learning disabilities (e.g., Dyslexia) impact program comprehension and learning.

References

1 Madeline Endres, Zachary Karas, Xiaosu Hu, Ioulia Kovelman, and Westley Weimer. *Relating Reading, Visualization, and Coding for New Programmers: A Neuroimaging Study*. International Conference on Software Engineering (ICSE), 2021.

2 Madeline Endres, Madison Fransher, Priti Shah, and Westley Weimer. *To Read or To Rotate? Comparing the Effects of Technical Reading Training and Spatial Skills Training on Novice Programming Ability*. Foundations of Software Engineering (ESEC/FSE), 2021.

3.10 Measuring Cognitive Effort During Programming: current methods and the cognitive offloading tools of the future

Sarah Fakhoury (Microsoft Research (MSR) – Redmond, US)

Programming tasks require inherent cognitive load, but the design of the tools and languages a programmer uses to complete their task can either increase mental burden, or optimize for it. I briefly talk about our work using simultaneous fNIRS and eye tracking to measure cognitive effort caused by programming antipatterns in the context of bug localization tasks. We observe that we cannot make assumptions about cognitive effort based on traditional metrics like correctness and time on task alone. Novel methods give us the ability to develop and test theories of how and why various factors influence comprehension.

Next I briefly touch on pain points related to tooling that the community can make joint strides in. Finally, I raise questions about the future of programming comprehension research in the age of AI coding assistants that aim to serve as cognitive offloading tools.

3.11 Sensing in the Wild: Increasing Productivity by Sensing Interruptibility

Thomas Fritz (Universität Zürich, CH)

The modern workplace is more demanding than ever before. Software developers have to work on a wide variety of cognitively demanding tasks, face constant context switches, work in distributed teams, and have blurred work-life boundaries. What does it mean to be productive in this context, and how can we best support developers in staying focused? To address these questions and better understand developers' cognitive and emotional states,

in our research, we employ a variety of sensors in a range of studies, from small controlled lab experiments to 8-week-long field studies. The results show that while it is not always feasible to gather fine-grained biometric data for highly accurate classifications, even with coarser-grained data, we can develop approaches that take into account developers' states and help boost their productivity.

3.12 Predicting human reading comprehension from eye movements

Lena A. Jäger (Universität Zürich, CH)

Joint work of Lena A. Jäger, David R. Reich, Silvia Makowski, Ahmed Abdelwahab, Niels Landwehr, Tobias Scheffer, Paul Prasse, Frank Goldhammer
Main reference David Robert Reich, Paul Prasse, Chiara Tschirner, Patrick Haller, Frank Goldhammer, Lena A. Jäger: "Inferring Native and Non-Native Human Reading Comprehension and Subjective Text Difficulty from Scanpaths in Reading", in Proc. of the ETRA 2022: Symposium on Eye Tracking Research and Applications, Seattle, WA, USA, June 8 – 11, 2022, pp. 23:1–23:8, ACM, 2022.
URL https://doi.org/10.1145/3517031.3529639

Eye movements in reading have long been known to reflect cognitive processes involved in reading. Abundant evidence from cognitive psychology and psycholinguistics demonstrates that, among many other factors, reading comprehension is a significant predictor of fixation durations. However, it has turned out that, conversely, predicting reading comprehension from eye movements is much more challenging. In my talk, I will present two different approaches to predict reading comprehension from eye-tracking data. I will first present a psychologically motivated generative model of eye movements in reading from which we derive a discriminative Fisher kernel to predict reading comprehension. Second, I will present a neural sequence model that processes raw scanpaths along with the read text and predicts the reader's comprehension. The proposed models outperform the previous state-of-the-art methods. Finally, I will discuss the current challenges that models aiming to predict reading comprehension from eye movements are facing.

References

1 David R. Reich, Paul Prasse, Chiara Tschirner, Patrick Haller, Frank Goldhammer, and Lena A. Jäger. *Inferring Native and Non-Native Human Reading Comprehension and Subjective Text Difficulty from Scanpaths in Reading*, ETRA 2022.
2 Silvia Makowski, Lena A. Jäger, Ahmed Abdelwahab, Niels Landwehr, and Tobias Scheffer. *A discriminative model for identifying readers and assessing text comprehension from eye movements*. ECML-PKDD 2018. In Brefeld et al. (eds). *Machine Learning and Knowledge Discovery in Databases*, Springer, Cham, Switzerland, 2019.

3.13 Investigating Programming Expertise With Event-Related Desynchronization

Timothy Kluthe (University of Nevada – Las Vegas, US)

Joint work of Igor Crk, Timothy Kluthe, Andreas Stefik
Main reference Igor Crk, Timothy Kluthe, Andreas Stefik: "Understanding Programming Expertise: An Empirical Study of Phasic Brain Wave Changes", ACM Trans. Comput. Hum. Interact., Vol. 23(1), pp. 2:1–2:29, 2016.
URL https://doi.org/10.1145/2829945

With the recent resurgence of interest in applying cognitive science technologies in the study of computer science, we present some of our previous research on the topic. Using electroencephalography and Event-Related Desynchronization measurements, we investigated several sub-bands associated with various cognitive subprocesses and how they differ in programmers of varying experience levels when working through programming comprehension tasks. Currently, we are working on making data science accessible for everyone. To achieve this goal, we will be gathering empirical evidence on design and syntax choices using typical human factors methodologies such as surveys and usability studies. In addition to this, we will be designing neuroscience-based studies which look at the same problems from a different angle.

3.14 Computational NeuroSE

Takatomi Kubo (Nara Institute of Science and Technology, JP)

Joint work of Takatomi Kubo, Takeshi D. Itoh, Yoshiharu Ikutani
Main reference Yoshiharu Ikutani, Takatomi Kubo, Satoshi Nishida, Hideaki Hata, Kenichi Matsumoto, Kazushi Ikeda, Shinji Nishimoto: "Expert Programmers Have Fine-Tuned Cortical Representations of Source Code", eNeuro, Vol. 8(1), Society for Neuroscience, 2021.
URL https://doi.org/10.1523/ENEURO.0405-20.2020

NeuroSE is a research field in software engineering (SE) that makes use of neuroscientific methods and knowledge to better understand the software development process, as well as the software system itself as the outcome of the process. The neuroscience is expected to contribute to a better understanding of the SE process and to affect the software system itself positively as a consequence. The NeuroSE field is characterized by collaboration of researchers from various disciplines, and still relatively young. In the next decade, NeuroSE should advance to the next stage. One of the missing pieces in the current NeuroSE is "computational approach".

In the neuroscience, computational neuroscience was advocated and is a branch of neuroscience which employs mathematical models, computer simulations and theoretical analyses with abstractions of the brain to understand the principles that govern the development, structure, and functions of the nervous system. In the history of computational neuroscience, David Marr offered a distinction of three levels: (i) computational theory, (ii) representation and algorithm, and (iii) hardware implementation.

These concepts should have high affinity to SE or its related fields since these terms are often used in them. From such background, the emergence of Computational NeuroSE should be natural. Computational NeuroSE will lead to unveiling the algorithm in the brain to understand the algorithms in the external world. I will also mention the potential interaction between Computational NeuroSE and AI4code/ML4code in the presentation.

3.15 Safe and Secure Software Engineering – A Program Comprehension Perspective

Jürgen Mottok (OTH Regensburg, Germany)

Joint work of Lisa Grabinger, Florian Hauser, Jürgen Mottok

What is the difference between experts and novices in software engineering disciplines like requirements engineering, analysis and design, and implementation? Functional Safety and IT-Security demand highly qualified software engineers with a deep conceptual understanding of e.g. strength and weakness of programming techniques.

We are pursuing different experimental settings, including different reading techniques, scaffolding approaches, mixed model approaches or EMME to evaluate which techniques are useful to guide the transformation process from a novice to an expert in software engineering.

We are interested in replication studies and can provide an Eye-Tracking laboratory with 14 Tobii Pro Spectrum (600Hz) for field studies.

3.16 The logical reasoning network encodes algorithms even in programming novices reading plain-language description of programming functions

Yun-Fei Liu (Johns Hopkins Univ. – Baltimore, US)

Joint work of Yun-Fei Liu, Marina Bedny
Main reference Yun-Fei Liu, Judy Kim, Colin Wilson, Marina Bedny: "Computer code comprehension shares neural resources with formal logical inference in the fronto-parietal network", eLife, Vol. 9, p. e59340, eLife Sciences Publications, Ltd, 2020.
URL https://doi.org/10.7554/eLife.59340

In a previous functional MRI study, we found the fronto-parietal logical reasoning network is engaged during code comprehension in programming experts. Additionally, we can use support vector machine (SVM) to classify FOR and IF algorithms using the spatial activation patterns in the regions in this network. In an ongoing project, we ask whether the fronto-parietal system processes the semantic content (i.e., the algorithms) regardless of the specific syntax in which the algorithms are presented – even in programming novices. During the MRI scan, Programming-naïve students read "pseudocode" passages, which are natural language descriptions of Python functions used in the previous expert study. Preliminary findings suggest that pseudocode reading also engaged the fronto-parietal logical reasoning network, and that FOR and IF algorithms (expressed with plain language rather than Python code) were also decodable in this network. Overall, the data suggest the logical reasoning system is recycled for code comprehension. This logical reasoning system represents algorithms all along, independent of the syntax of specific programming languages, and even in individuals with 0 programming experience.

3.17 An Eye Tracking Analysis of Tracing and Debugging Collaboration among Programming Pairs

Maria Mercedes T. Rodrigo (Ateneo de Manila University – Quezon City, PH)

Joint work of Maria Mercedes T. Rodrigo, Maureen Villamor
Main reference Maureen Villamor, Ma. Mercedes T. Rodrigo: "Gaze collaboration patterns of successful and unsuccessful programming pairs using cross-recurrence quantification analysis", Res. Pract. Technol. Enhanc. Learn., Vol. 14(1), p. 25, 2019.
URL https://doi.org/10.1186/s41039-019-0118-z

We make use of Cross-Recurrence Quantification Analysis (CRQA) to characterize tracing and debugging collaboration behaviors among programming students engaged in a pair programming task. We describe how successful and unsuccessful pairs significantly differ in their gaze patterns. We also describe how prior knowledge and acquaintanceship affect pair success.

References

1 Maureen Villamor, Ma. Mercedes T. Rodrigo: *Gaze collaboration patterns of successful and unsuccessful programming pairs using cross-recurrence quantification analysis*. Res. Pract. Technol. Enhanc. Learn. 14(1): 25 (2019)
2 Maureen Villamor, Ma. Mercedes T. Rodrigo: *Predicting Successful Collaboration in a Pair Programming Eye Tracking Experiment*. UMAP (Adjunct Publication) 2018: 263-268

3.18 Exploring Common Code Reading Strategies in Debugging

Maria Mercedes T. Rodrigo (Ateneo de Manila University – Quezon City, PH) and Christine Lourrine S. Tablatin (Pangasian State University, PH and Ateneo de Manila University – Quezon City, PH)

Main reference Christine Lourrine S. Tablatin, Maria Mercedes T. Rodrigo: "Identifying Code Reading Strategies in Debugging using STA with a Tolerance Algorithm", APSIPA Transactions on Signal and Information Processing, Vol. 11(1), pp. –, 2022.
URL https://doi.org/10.1561/116.00000040

The purpose of this study was to identify the common code reading strategies of the high and low performing students engaged in a debugging task. Using Scanpath Trend Analysis (STA) with a tolerance on eye tracking data, common scanpaths of high and low performing students were generated. The common scanpaths revealed differences in the code reading patterns and code reading strategies of high and low performing students. High performing students follow a bottom-up code reading strategy when debugging complex programs with logical and semantic errors. A top-down code reading strategy is employed when debugging programs with simple control structures, few lines of code, and simple error types. These results imply that high performing students use flexible debugging strategies based on the program structure. The generated common scanpaths of the low performing students, on the other hand, showed erratic code reading patterns, implying that no obvious code reading strategy was applied. The identified code reading strategies of the high performing students could be explicitly taught to low performing students to help improve their debugging performance.

3.19 Detecting Expertise in Developer Eye Movements

Bonita Sharif (University of Nebraska – Lincoln, US)

Joint work of Bonita Sharif, Salwa Aljehane, Jonathan Maletic
Main reference Salwa Aljehane, Bonita Sharif, Jonathan I. Maletic: "Determining Differences in Reading Behavior Between Experts and Novices by Investigating Eye Movement on Source Code Constructs During a Bug Fixing Task", in Proc. of the 2021 Symposium on Eye Tracking Research and Applications, ETRA 2020, Virtual Event, Germany, May 25-27, 2021, Short Papers, pp. 30:1–30:6, ACM, 2021.
URL https://doi.org/10.1145/3448018.3457424

What constitutes developer expertise? Could expertise be determined based on the nature of the task rather than by how many years a developer worked in the field? It is also more likely that expertise is not necessarily a binary decision: expert vs. non-expert, as there may be variations of expertise. We start to address these questions by investigating developer expertise prediction solely from a biometric data source namely, eye fixation data on source code elements. Which clustering similarity metrics work best for determining developer expertise on eye fixation sequences? What should the level of granularity be when looking at fixations and their sequences? Can we train a model to predict expertise with high confidence given eye tracking data for a particular task? One problem facing this research is the lack of enough eye tracking datasets on a variety of tasks from a diverse demographic. Another issue is the individual differences that exist even in how two experts solve a task. Given this, we are seeking to uncover some commonalities in how people read and navigate between code chunks/beacons when they have similar expertise.

References

1 Salwa Aljehane, Bonita Sharif, Jonathan I. Maletic: *Determining Differences in Reading Behavior Between Experts and Novices by Investigating Eye Movement on Source Code Constructs During a Bug Fixing Task.* ETRA Short Papers 2021: 30:1-30:6
2 Naser Al Madi, Cole S. Peterson, Bonita Sharif, Jonathan I. Maletic: *From Novice to Expert: Analysis of Token Level Effects in a Longitudinal Eye Tracking Study.* ICPC 2021: 172-183

3.20 How does the Brain Change during Programming Learning?

Janet Siegmund (TU Chemnitz, DE)

Learning programming has been a challenge for decades, despite many approaches to support students in mastering this important skill. The neuro-cognitive perspective of programming has shown that language-processing skills are essential during programming. Thus, we will be looking into how tapping into language learning can support students who are learning programming. To this end, we teach students an artificial language before they learn programming and evaluate whether this improves their performance. This additional step can be one piece of the puzzle to teach programming to everyone. In a long-term study, we want to observe how the programming skills of students evolve and how that will be reflected in their neuronal representation of programming. Similar to other skills that have an efficient neuronal representation with a high level of expertise, we evaluate whether we can find such a similar change for programming skills.

3.21 Evidence-Based Programming and the "Quorum Project"

Andreas Stefik (University of Nevada, Las Vegas, US)

In this talk, we will explore the Quorum project, an attempt to make the design programming languages more evidence based in regard to their human factors impact. In the process, we will discuss the history of evidence gathering, competing language designs, and several studies that document the impact of such designs on people at various ability levels and in different demographics

3.22 Making Novices More Like Experts?

Westley Weimer (University of Michigan – Ann Arbor, US)

Joint work of Endres, Madeline; Huang, Yu; Leach, Kevin
Main reference Madeline Endres, Zachary Karas, Xiaosu Hu, Ioulia Kovelman, Westley Weimer: "Relating Reading, Visualization, and Coding for New Programmers: A Neuroimaging Study", in Proc. of the 43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021, pp. 600–612, IEEE, 2021.
URL https://doi.org/10.1109/ICSE43902.2021.00062

Can we make novices more like experts through targeted training or neurostimulation? We discuss investigations of code comprehension, data structures, code writing and code review, including contextual and functional connectivity analyses. We conclude with a call to arms about the potential use of transcranial magnetic stimulation for causal analyses and behavioral improvements.

3.23 Could we please be a bit more explicit?

Marvin Wyrich (Universität Stuttgart, DE)

Main reference Marvin Wyrich, Justus Bogner, Stefan Wagner: "40 Years of Designing Code Comprehension Experiments: A Systematic Mapping Study", arXiv, 2022.
URL https://doi.org/10.48550/ARXIV.2206.11102

We looked at the study designs of code comprehension experiments from the past 40 years and summarized them in a systematic mapping study. We noticed that the primary studies do not yet name, define, and explain too well what construct they actually intend to measure. Before we go after code comprehension with new methods, perhaps we should pause for a moment and clarify whether we intend to measure the same thing. What could code comprehension be?

4 Working Groups

4.1 Eye Tracking Best Practices & Ideas

Teresa Busjahn (Hochschule für Technik und Wirtschaft Berlin – Berlin, DE)
Martha E. Crosby (University of Hawaii at Manoa – Honolulu, US)
Maria Mercedes T. Rodrigo (Ateneo de Manila University – Quezon City, PH)
Christine Lourrine S. Tablatin (Pangasian State University, PH)
Westley Weimer (University of Michigan – Ann Arbor, US)

This working group discussed the current state of the art of eye tracking use for research in computing and gaps in the literature and eye tracking challenges.

4.1.1 Discussed Open Problems

- Programming-specific methodological innovation: The working group members discussed how elements of programming (e.g., navigating through multiple files) make some standard eye tracking approaches more challenging (e.g., static stimuli). The members of the working group agreed that there is currently a gap between the experimental capabilities of eye tracking and studying professional software development. As summarized by Dr. Crosby, "the bigger problems are too big. The experiments we can do [with eye tracking] are code snippets, not production code".
- Individual differences and generalization: Working group members report that it is challenging to account for individual differences in eye tracking studies and to have confidence that results may generalize. As Dr. Busjahn stated, "Individual differences are a big issue. What's hard for me might not be hard for you (e.g., if you've seen the algorithm before)." Additionally, working group members noted that individual differences can have a large impact on eye tracking results; even the amount of coffee you had while practicing vs. when participating in the study may matter.
- Recruitment: Several working group members noted that they find it difficult to recruit participants for eye tracking studies. Group members noted that for researchers in academia, it is often impossible to pay professional developers their standard hourly rate to participate in eye tracking studies, making the recruitment of professional developers challenging. Additionally, group members note that it can be difficult to recruit diverse populations of programmers, something that is especially important in eye tracking research as demographic factors can influence the results (e.g., if your native language is read from right to left or left to right).
- Standardization of analysis methodologies and empirical results reporting: All of the working group members agree that more standardization is needed for conducting, analyzing, and presenting eye tracking results in computer science research. In particular, concerns were raised regarding the use of parametric statistical tests when not appropriate or the lack of multiple comparison correction in some eye tracking studies.

4.1.2 Possible Approaches and Recommendations

For the problems discussed above, the members of this working group agreed on a set of possible approaches, recommendations, and talking points.

1. *Experimental Design:* Use an established metric (and established name for it) if you can (e.g., from the Holmqvist et al. book [1]). In addition, most CS papers using eye tracking have individual participants complete tasks that are too different. Avoid! "Think before you start the eye tracker."
2. *Statistical Transparency:* Most CS papers using eye tracking do not indicate which tests they use (e.g., to assess normality of data and otherwise check assumptions). In addition, always report on basic measures (like fixation duration) even if you're not using them statistically.
3. *Statistical Rigor:* Most CS papers using eye tracking do not check to see if parametric tests are appropriate. In addition, most papers do not consider false discovery rate or correcting for multiple comparisons. Be intentional and use non-parametric tests and correct for multiple comparisons when needed!
4. *Between-Subjects Comparisons:* Most CS papers using eye tracking do not correctly handle normalization.
5. *Good Starting Point Recommendations:*
   - eyecode (for automated AOI analysis for code and prose)
   - code2vec (for salient points in the code)
   - pre-testing for stimuli with a similar population to assess times and difficulty
   - stick to one specific thing to measure, resist the temptation to have five conditions (esp. given the noisy nature of eye tracking)
   - the guide on eye tracking studies in software engineering by Sharafi et al. [2] also provides valuable cues
6. *Anonymous Recommendation:* SE conference program committees should have an 'on-staff' person who has done behavioral science (social science) research to help assess claims. This person doesn't have to write anything unless they spot something. We who are supervising students need to ensure that they have the knowledge.
7. *Longer-Term Question:* Do we have reason to suspect that our results will differ across populations?
   - Examples: Left-to-Right vs. Right-to-Left reading order, dyslexia, corrective lenses, shape of the eye (which may correlate with race/ethnicity), spatial ability.
   - However, we should be careful to consider individual differences: more variability in outcomes comes from the person, not the group membership.

4.1.3 Conclusions

Overall, the working group agrees that while eye tracking can be a powerful tool to study program comprehension, there remain many challenges facing the use of this technology in various research contexts. To help address these challenges, the members of this working group discussed the state of the art and recommend best practices to improve research quality and result standardization.

References

1 Kenneth Holmqvist, Marcus Nystrom, Richard Andersson, Richard Dewhurst, Halszka Jarodzka, Joost Van De Weijer; Eye Tracking: *A comprehensive guide to methods and measures*. Oxford University Press, 2015

2 Zohreh Sharafi, Bonita Sharif, Yann-Gaël Guéhéneuc, Andrew Begel, Roman Bednarik, Martha E. Crosby: *A practical guide on conducting eye tracking studies in software engineering*. Empirical Software Engeeri g 25(5): 3128-3174 2020.

4.2 Readability

Andrew Begel (Carnegie Mellon University – Pittsburgh, US)
Annabelle Bergum (Universität des Saarlandes – Saarbrücken, DE)
Madeline Endres (University of Michigan – Ann Arbor, US)
Sarah Fakhoury (Microsoft Research (MSR) – Redmond, US
Timothy Kluthe (University of Nevada – Las Vegas, US)
Yun-Fei Liu (Johns Hopkins Univ. – Baltimore, US)
Bonita Sharif (University of Nebraska – Lincoln, US
Jan Stelovsky (University of Hawaii at Manoa – Honolulu, US)
Marvin Wyrich (Universität Stuttgart, DE)

This working group focused on defining *readability* in the context of code comprehension and programming human studies. To this end, the working group first brainstormed various aspects of readability, coming up with a preliminary taxonomy of related concepts and factors. In particular, the working group considered the definition of code readability (e.g., is readability the same thing as understandability or traceability), the impact of context and programming task on readability (e.g., code review, debugging, API use), metrics for measuring readability (e.g., subjective vs. objective metrics, binary vs. continuous), and potential experimental designs for readability studies (e.g., ecological validity, participant demographics impacts).

Following this brainstorming session, the working group discussed open questions and future directions concerning code readability, ultimately producing a list of 27 research questions of interest for the community.

4.2.1 Readability Open Questions and Research Directions

- *Foundational and Definitional Questions*
  1. How is readability connected to other foundational terms in code research such as comprehension, debuggability, usability, traceability, complexity, or maintainability?
  2. What does it mean to be a good code reader?
  3. How fast does the average developer read code?
  4. Are faster readers better developers?
  5. What do professional developers think readability is?
  6. Is there a trade off between code readability, and code conciseness or code quality?
  7. Does readable code have to be slower?
- *Participant Demographics and Differences*
  1. How do developer demographic factors correlate with or impact code readability?
  2. How is readability impacted when comments and documentation are in a programmer's native language?
  3. Does native natural language effect readability?
  4. How does anticipated readability impact comprehension (e.g., self-efficacy)?
  5. How well do metrics for sight reading code work for hearing code (e.g., for blind and visibly impaired programmers)?
- *Readability, Biometrics, and Cognition*
  1. Does readable code help you use less cognitive effort?
  2. How does program difficulty impact brain activity when reading code?
  3. Does reading other languages or music transfer to code reading?

- *Study Design and Metrics*
  1. How does the study task affect comprehension process or readability performance?
  2. Should your readability metrics be customized for the task, experimental context, or participants? Are they robust to comprehension strategy?
  3. How fast can you measure code readability?
- *Potential Study Tasks*
  1. How does code presentation affect readability?
  2. How do syntactic features of code affect readability?
  3. Do formal style guides or formatting rules make code more readable?
  4. What mechanisms help make difficult code easier to read (e.g., interactive tools in code editors)?
  5. How do comments affect code readability?
  6. Can two people read code better than one?
  7. Do many eyes make bugs shallow?
  8. How does readability of code or data affect the effectiveness of debugging?

4.2.2 Conclusions

Overall, everyone in the working group agrees that program readablity is an important research concept to consider in future research. However, there is also a consensus that more work needs to be done as a community to define readability and understand how it contrasts, overlaps, or connects with other core concepts in program comprehension research such as understandability and debuggability. There was also the acknowledgement of the need for a literature review of work relating to code readability to help build consensus in the research community.

4.3 Statistics and machine learning to predict and model program comprehension

Sven Apel (Universität des Saarlandes – Saarbrücken, DE)
André Brechmann (Leibniz-Institut für Neurobiologie – Magdeburg, DE)
Janet Siegmund (TU Chemnitz, DE)
Lena Jäger (Universität Zürich, CH)
Andreas Stefik (University of Nevada, US)
Takatomi Kubo (Nara Institude of Science and Technology, JP)

In this working group, the members discussed how the program comprehension research community could use statistics and machine learning combined with insights from psychology and cognitive science to predict and model program comprehension.

4.3.1 Discussion Summary

One foundational problem discussed by the working group is what is meant by a model of program comprehension. For example, is it a collection of inputs and outputs, such as a model that represents the activation in the brain when a particular stimulus is provided to a person?

The working group members note that one very important distinction is that when using linear models or other traditional machine learning techniques, we need specific features for code comprehension. These features would depend on the experimental designs used. Additionally, the members note that it is important that when modeling program comprehension, researchers should keep in mind the bigger picture and the impact of this work. For example, researchers should consider what *actions* to take or not take depending on the predictions of such a model (e.g., if a model predicts X about program comprehension, should practitioners change programming languages, computer science curricula, or something else?).

4.3.2 Open Questions and Suggestions

Beyond general discussion, the working group discussed open questions regarding what a model of program comprehension should look like. The questions the working group recommends the community to consider include:

- What should these models of program comprehension predict?
  - Suggested outputs include programmer time or accuracy on a task, programmer neural response, eye movement, or the the overall impact of various programming language features (e.g., static typing, or syntax choices).
- What inputs should go into program comprehension models?
  - Suggested inputs include brain responses (e.g., fMRI, EEG), eye tracking information, response time and accuracy data, field data about programming usage in practice, corrective information, source code features, and subjective ratings about code or difficulty.
- What demographics should these program comprehension models be predictive for?
- How should we test program comprehension models?
- What is the societal impact of models of program comprehension?
  - Suggested impacts to consider include changing programming languages or predicting how they should change, and educational implications.
- How should we define a scientific process whereby we can create and validate such a model of program comprehension?
  - Suggestions from the working group members include taking existing experiments with known results and vary them based on a proposed model or breaking down the model in terms of basic operations of program comprehension. However, the working group members also note that the basic operations of program comprehension are not yet well defined.

4.3.3 Conclusions

The working group members suggest that as a community, we should decide what the goal of creating a model of program comprehension. Based on that, it will be possible to decide what kind of model to pursue. Additionally, the working group members discussed what existing cognitive models from other fields program comprehension researchers can build on. One such model could be *predictive coding* [1]. Language models ext on predictive coding that have been trained on several programming languages (such as C or Java). Other models from language that have a strong basis in psychology that could be interesting to consider for code include the ACT-R model of sentence processing [2].

References

1 Linda Ficco, Lorenzo Mancuso, Jordi Manuello, Alessia Teneggi, Donato Liloia, Sergio Duca, Tommaso Costa, Gyula Zoltán Kovacs, Franco Cauda; *Disentangling predictive processing in the brain: a meta-analytic study in favour of a predictive network*. Nature, 2021.

2 Lewis, R. L., Vasishth, S. (2013). An activation-based model of sentence processing as skilled memory retrieval. In Cognitive Science, 29(3), 375-419

5 Open problems

At the end of the seminar, all participants discussed open problems facing program comprehension research. One common theme that emerged was the need for clarity of definitions and experimental dimensions in the field. Many participants viewed doing so as a first step towards unifying as a research community and identifying the most important outstanding directions and questions for program comprehension researchers. As a result, the participants of the seminar worked together to build a preliminary list of concepts for an actionable taxonomy of program comprehension.

5.1 A Taxonomy of Program Comprehension

Participants were generally in consensus that the field of program comprehension currently lacks an explicit taxonomy codifying the research space. As a result, participants worked together to brainstorm categories and dimensions that should be included in such a taxonomy. The goal of this taxonomy was to give researchers a practical guide of dimensions to consider when designing a program comprehension study. The discussion was moderated by Dr. Andrew Begel. In the rest of this section, we present the preliminary taxonomy categories determined by the seminar participants.

- *Level of Comprehension:* Letters, Lexemes, Words, Program structure, Program semantics, Program intent, Program rationale
- *Reason for Comprehension:* Design, Knowledge retention, Immediate use, Understanding
- *Comprehension Success Criteria:* How well is the intent or idea of some code transferred to its audience? (e.g., more formally, code comprehension has succeeded when *Comprehension(Coding(original_idea)) = original_idea*)
- *Code Precision:* Pseudocode, Natural Language, Primitive Language, Symbolic Programming Language (like PERL or APL), Math
- *Amount, Type, or Organization of Code:* Code, Program, Snippet, Architecture, Software, Docs, Modules, Functions, Libraries, Comments, Error messages, Logs
- *Domain of the Code:* Production Software, Experimental or Prototype, Educational, Scientific Computing, Data Processing, Games, Art, Business, Systems, Security, Etc.
- *Code Language Aspects:* Templates or Generics, Classes, Strong or Dynamic Types
- *Modality:* Physical Code Representation (e.g., visual, audio, tactile, features to support presentation such as syntax highlighting), Mental Model Representation
- *Presentation (can do any of these as any modality):* Text, Graph, Data Structure Graph, Blocks, Flow Charts
- *Experimental Task:* Reading, Writing, Explaining (consider the target audience – see code as a boundary object below), Communicating, Teaching, Tracing, Debugging, Reasoning about variable values, Dependencies, Judging design quality, Refactoring, Fixing Bugs or

Security Vulnerabilities, Adding Features, Code Review, Revising, Code Summarization, Writing Documentation
- *Experimental Interventions (Real time):* Syntax highlighting, Font, Heatmap, Define/Use/ Navigate, Structural Elision Collapse / hide parts of code, Provided code summary, code layout, error messages
- *Experimental Time Scale:* Session, Sprint, Milestone, Day, Week, Semester
- *Code as a Boundary object:* Between coder and computer, Between coder and coder, Between coder and tester, Between coder and user, Between coder and boss, Between student and teacher, Between computer and computer, Between coder and generic outside audience, Size of the audience (either code readers or code users), etc.
- *Number of Code Readers:* Solo, Pair, Mob, Team, Cohort, Replacement
- *Simultaneity:* Interactivity, Synchronous Communication, Asynchronous Communication, No access to code author, No access to friends
- *Person or Programmer In Your Study:* Demographics (e.g., gender, ethnicity, cultural background), Skills or Expertise, Job or Industry, Psychometric tests, Native natural languages, Cognitive difference (e.g., neurodiversity, spatial reasoning ability), Physical Disability, Pregnancy Brain, Mental Health, Identity
- *Experimental Metrics:* Time on task, Accuracy, Speed, Reading Distribution – Spatial/temporal (e.g., fault localization for debugging), Subjective ratings / self reporting, Perceived difficulty or understanding, Continuous or discrete, Cost, Affect, Motivation, Self-efficacy, Biometric and cognitive metrics (e.g., focus, cognitive load, gaze path, brain activity),
- *Code Metrics:* Code complexity Size Debuggability Maintainability Readability
- *Experimental Hypothesis:* from a given hypothesis, you can choose different elements from various categories in the taxonomy.

5.1.1 Conclusions

In this seminar, the participants proposed preliminary categories for a taxonomy of program comprehension research. We hope that the community will build on this work to create a more complete and formal taxonomy going forward.

Participants

- Sven Apel
Universität des Saarlandes – Saarbrücken, DE
- Andrew Begel
Carnegie Mellon University – Pittsburgh, US
- Annabelle Bergum
Universität des Saarlandes – Saarbrücken, DE
- André Brechmann
Leibniz-Institut für Neurobiologie – Magdeburg, DE
- Teresa Busjahn
Hochschule für Technik und Wirtschaft Berlin – Berlin, DE
- Martha E. Crosby
University of Hawaii at Manoa – Honolulu, US
- Sarah D'Angelo
Google – Mountain View, US
- Madeline Endres
University of Michigan – Ann Arbor, US
- Sarah Fakhoury
Microsoft Research (MSR) – Redmond, US
- Thomas Fritz
Universität Zürich, CH
- Lena A. Jäger
Universität Zürich, CH
- Timothy Kluthe
University of Nevada – Las Vegas, US
- Takatomi Kubo
Nara Institute of Science and Technology, JP
- Yun-Fei Liu
Johns Hopkins Univ. – Baltimore, US
- Jürgen Mottok
OTH Regensburg, Germany
- Maria Mercedes T. Rodrigo
Ateneo de Manila University – Quezon City, PH
- Bonita Sharif
University of Nebraska – Lincoln, US
- Janet Siegmund
TU Chemnitz, DE
- Andreas Stefik
University of Nevada – Las Vegas, US
- Jan Stelovsky
University of Hawaii at Manoa – Honolulu, US
- Christine Lourrine S. Tablatin
Pangasian State University, PH
- Westley Weimer
University of Michigan – Ann Arbor, US
- Marvin Wyrich
Universität Stuttgart, DE

Remote Participants

- Andrew Duchowski
Clemson University, US
- Russell Poldrack
Stanford University, US