UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Faculty of Computer Science, Electrical Engineering
and Mathematics

Dissertation

Empirical Analysis of Eye Movements during Code Reading: Evaluation and Development of Methods

by

Teresa Busjahn

April 2021

Abstract. Code reading is a fundamental part of program comprehension. While studying eye movements has provided valuable insights into natural-language text comprehension for decades, its application in program comprehension is fairly recent and brings about many methodological challenges. This work evaluates and adapts existing methodological approaches, resulting in a customized event detection and a novel correction for spatial error.

The suitability of the proposed methodology is demonstrated with two exemplary research questions. First, it is analyzed whether natural-language text and source code are read differently. The second question concerns differences in how novices and experts read programs. To obtain a comprehensive picture of code reading, a wide range of established as well as specifically devised measures is used for analysis, e.g. sequence alignment is adapted into an instrument for examining reading approaches.

In some regards novices already read source code differently than natural-language text, for experts the differences are even more pronounced. Participants look at a greater proportion of natural-language text than of source code. Novices partly exhibit a different code reading behavior than experts, e.g. the latter attend to the main-method much sooner.

Eye movements provide manifold information to deepen the understanding of code reading. The developed methodology can be applied to many questions in software engineering and programming education.

Zusammenfassung. Das Lesen von Quelltext ist ein essentieller Teil des Programmverstehens. Während die Analyse von Blickbewegungen seit Jahrzehnten wertvolle Einblicke in das Verstehen von natürlichsprachlichem Text liefert, ist sie beim Programmverstehen relativ neu und es ergeben sich eine Reihe methodischer Probleme. Diese Arbeit evaluiert und adaptiert methodische Ansätze, so werden u.a. eine spezifische Ereignisdetektion und ein neues Korrekturverfahren für räumliche Fehler entwickelt.
Die Eignung der beschriebenen Methodik wird mit zwei exemplarischen Forschungsfragen demonstriert. Zunächst wird analysiert, ob natürlichsprachlicher Text und Quelltext unterschiedlich gelesen werden. Die zweite Frage betrifft die Unterschiede zwischen Anfänger:innen und Expert:innen. Für ein umfassendes Bild wird ein breites Spektrum etablierter als auch speziell entwickelter Analysemaße eingesetzt, z.B. Sequenzalignment zur Beschreibung von Leseansätzen.
Bereits Anfänger:innen lesen Quelltext teilweise anders als natürlichsprachlichen Text, bei Expert:innen sind die Unterschiede noch ausgeprägter. Bei natürlichsprachlichem Text werden mehr Bereiche betrachtet als bei Programmen. Anfänger:innen lesen Quelltext zum Teil anders als Expert:innen, so befassen sich letztere viel früher mit der main-Methode.
Blickbewegungen liefern reichhaltige Informationen über das Quelltextlesen. Die entwickelte Methodik lässt sich auf viele Fragen im Bereich der Softwaretechnik und der Programmierausbildung anwenden.

To my family

A huge thank you to my advisor Carsten Schulte for letting me pursue my research interests and all those great conversations. I'm also grateful to my reviewers and committee members Bonita Sharif, Erik Barendsen, Sascha Tamm, Johannes Magenheim, and Harald Selke. Sascha Tamm additionally provided valuable guidance throughout the work on this thesis.

Thanks also to everyone else who contributed to this work, of which I'd like to mention in particular (in alphabetical order): Adrian Voßkühler, Andrew Begel, Bonita Sharif, Hana Vrzakova, Michael Hansen, Paul Orlov, Roman Bednarik, and Sebastian Lohmeier. I'd also like to thank the participants of my study and of the first EMIP workshops for their input. Furthermore, I greatly enjoyed the exchange with colleagues at Freie Universität Berlin, University of Applied Sciences Berlin, and University of Eastern Finland.

I could not have accomplished this work without the constant support of my husband Rüdiger, my family, especially my dad, and Inga Semmler. Finally, I'd like to mention that my son is awesome.

Contents

1 Introduction

1.1 Motivation

1.1.1 The importance of code reading and understanding

Program comprehension plays an integral role in software engineering as well as in programming education. It takes up a considerable part of the time and mental resources of programmers. Understanding source code includes the perception of its text, extracting information about the execution and algorithmic idea, and the construction of a mental representation using prior knowledge. Reading and understanding a source code are highly intertwined. When reading, the eyes do not merely move over the text to identify written characters, but due to the limited capacity of the working memory, information processing and filtering already occur during reading, so only information deemed relevant will be processed further. Furthermore, decisions are taken about where to focus next. Thus, reading is the first step in program comprehension [Bente, 2004, 309], [Busjahn et al., 2011, 1], [Busjahn & Schulte, 2013, 3,4,9], [Crosby et al., 2002, 58], [Deimel & Naveda, 1990, 1,5,7], [Dubochet, 2009, 177] , [Guéhéneuc, 2006, 1], [Pennington, 1987, 296], [Schulte, 2007, 307-313], [Schulte, 2008a, 150-152], [Schulte et al., 2010, 69,70], [Sharif & Shaffer, 2015, 807], [Spinellis, 2003a, xxii,1].[1]

Reading can be defined as "the ability to extract visual information from the page and comprehend the meaning of the text" [Rayner et al., 2012, 19]. Even though the work of Rayner et al. [2012] on the psychology of reading is mainly concerned with "normal" silent reading of natural-language text, this description also applies to source code. Code reading and comprehension cannot be separated and many program comprehension models draw on empirical studies on programmers carrying out reading tasks [Busjahn & Schulte, 2013, 6]. The terms code *reading* and *understanding* are even occasionally used interchangeably, sometimes an additional distinction is made to *tracing*, i.e. following the execution as opposed to understanding the algorithmic purpose of the code. This work focuses on the process of reading, not its product, the resulting mental representation. While comprehension is an internal cognitive process, reading has a physical component that can be observed and measured, e.g. with eye tracking [Busjahn & Schulte, 2013, 6], [Raymond, 1991, 8]. The term *code* reading instead of *program* reading is used to encompass source code in all stages, single syntax constructs, code snippets, or complete programs of any size.

[1] For an overview of models that aim to conceptualize program comprehension, see Mayrhauser & Vans [1994], Schulte et al. [2010], and Storey [2006].

Code reading is a very frequent activity during programming, which occurs in various contexts and at many stages in the software life cycle. Rooksby et al. [2006, 210] even call it "an unavoidable feature of programming work". Nevertheless, programming education and software engineering largely focus on writing and code reading is hardly brought up in literature and research. As a consequence, little is known about how programmers actually read. Furthermore, the skill of reading is often taken for granted in programming. Deeper knowledge about how programmers read source code can serve to improve teaching as well as support professional programmers [Aschwanden & Crosby, 2006, 6], [Busjahn & Schulte, 2013, 3,4,9,10], [Busjahn et al., 2014c, 3,8,9], [Busjahn et al., 2015a, 263,264], [Crosby & Stelovsky, 1990, 24], [Deimel Jr., 1985, 5], [Deimel & Naveda, 1990, 1,5,6], [Kölling & Rosenberg, 2001, 34], [Mannila, 2007, 140], [Nelson et al., 2017, 42,43], [Rooksby et al., 2006, 210], [Schulte, 2007, 307], [Sharif & Shaffer, 2015], [Spinellis, 2003a, xxi,1], [Xie et al., 2019, 12].

Programming tasks in which code reading is vital include writing code, debugging, modifying, and extending programs, testing, and code reviews. Especially for maintenance, reading can be considered a key activity, since despite the existence of design documents, it is necessary to understand existing code in order to find a suitable location for a modification, be it to correct errors, extend, or adapt the program. Code is read while under construction, but also after delivery. Programmers read by themselves as well as collaboratively, they read their own code, as well as that from others, and they read with varying goals, e.g. complete understanding or finding a bug. For example, one's own code is read during actively writing and refining a program, while debugging, understanding compiler messages, and to refresh details after taking a break. Programmers externalize their thoughts into source code and later connect different ideas through reading. Thus in programming, writing and reading are strongly interwoven. Code from others is read for maintenance and review, but also for professional development or when looking for a design for a particular problem. During pair programming one programmer often reads code while the other writes it. Additionally, code by colleagues or fellow students is read in order to help overcoming programming difficulties. Reading code from others, e.g. in program libraries or repositories, is also an opportunity to deepen the understanding of the craft and improve skills [Busjahn & Schulte, 2013], [Crosby et al., 2002, 58], [Deimel Jr., 1985, 5-8], [Deimel & Naveda, 1990, 5], [Fan, 2010, 2,7,11], [Mayrhauser & Vans, 1994, 44,45], [Raymond, 1991, 3], [Rooksby et al., 2006, 198,199,202,208], [Schulte, 2007, 307,317], [Schulte, 2008a, 150], [Spinellis, 2003a, xxii,1-10,17], [Spinellis, 2003b, 85-88], [Uwano et al., 2007, 2290].

Just as natural-language documents, source code can serve as memory artifact. By means of code, knowledge can be stored for later use and transferred between programmers [Dubochet, 2009, 176]. Rooksby et al. [2006] present an ethnomethodological study on reading during software development. By observing and interviewing professional programmers during their normal everyday work at a software company, they detected a multitude of situations in which reading occurred. Besides the already mentioned activities like writing code, debugging, and testing, they also found that code is read when programmers are searching the internet, reading emailed information, reading from textbooks, writing documentation, and sharing information on whiteboards.

Reading source code is also essential when learning a programming language, regardless of whether it is the first or an additional one. Programming requires a lot of predefined knowledge that can be acquired by reading, e.g. when new elements of a language, library or API are introduced, but also good programming style and standard solutions to programming problems. Learners read examples presented by the teacher and in textbooks, as well as code from fellow students. Reading programs has an educational value both for students and professional programmers, since good example code allows to learn syntax as well as good coding style [Busjahn & Schulte, 2013], [Campbell & Bolker, 2002, 23], [Crosby et al., 2002, 58], [Deimel Jr.,

1985], [Deimel & Naveda, 1990, 5,6,27], [Kimura, 1979], [Kölling & Rosenberg, 2001, 34], [Lister et al., 2004, 137], [Spinellis, 2003a, 2-6]. There are also a variety of learning and diagnostic tasks, which require code reading [Busjahn & Schulte, 2013, 9,10], [Deimel Jr., 1985], [Deimel & Naveda, 1990, 29-47], [Lister et al., 2004], [Mannila, 2007] (see 1.1.2 *Code reading in computer science education* for further discussion of this topic).

Programming requires a set of diverse skills, of which the ability to read and understand a program is a crucial one [Busjahn & Schulte, 2013, 3,7,9], [Deimel Jr., 1985, 5], [Deimel & Naveda, 1990, 1,21,22], [Kimura, 1979], [Spinellis, 2003a, xxii], or as an IT-security consulting service puts it: "Reading source code is like the x-ray goggles of hacking. The more you are able to see, the more bugs may appear under the hood" [Recurity Labs, 2020]. Spinellis [2003a, 3] encourages programmers to study existing code as a way to advance programming skills and points out that thanks to open-source software there is ample reading material [Spinellis, 2003b, 86]. A competent code reader probably possesses several different reading strategies, depending on the purpose [Deimel Jr., 1985, 6-8], [Deimel & Naveda, 1990, 11-14], [Schulte et al., 2010, 83]. Skilled code reading is especially relevant for maintenance and poor reading skills can result in serious financial consequences. If the effort to sufficiently understand a piece of existing code is too high, the respective software component sometimes ends up being discarded or rewritten [Deimel Jr., 1985, 5], [Dubochet, 2009, 177], [Fan, 2010, 7,11]. Reading competence also allows to better find and evaluate code that can be reused or adapted to bypass some coding altogether. Reading code can also provide examples of practices to avoid [Spinellis, 2003a, 3,9]. Partly it is also important to know what part of a program not to read [Raymond, 1991, 4]. Reading is a very effective way to find problems in code and the vast majority of errors can be ousted by inspection, better than by testing. However, code is mainly read in order to add functionality, modify existing features, adapt it to new environments and requirements, or for refactoring, not to fix errors [Spinellis, 2003a, xxii,7], [TechWell Contributor, 2001].

In the foreword to Spinellis' book "Code Reading: The Open Source Perspective", Dave Thomas aptly further illustrates the importance of code reading:

> "[T]he way to learn to write great code is by reading code. Lots of code. High-quality code, low-quality code. Code in assembler, code in Haskell. Code written by strangers ten thousand miles away, and code written by ourselves last week. Because unless we do that, we're continually reinventing what has already been done, repeating both the successes and mistakes of the past.
>
> I wonder how many great novelists have never read someone else's work, how many great painters never studied another's brush strokes, how many skilled surgeons never learned by looking over a colleague's shoulder, how many 767 captains didn't first spend time in the copilot's seat watching how it's really done.
>
> The irony is that there's never been a better time to read code. Thanks to the huge contributions of the open-source community, we now have gigabytes of source code floating around the 'net just waiting to be read. Choose any language, and you'll be able to find source code. Select a problem domain, and there'll be source code. Pick a level, from microcode up to high-level business functions, and you'll be able to look at a wide body of source code." [Spinellis, 2003a, xxi]

1.1.2 Code reading in computer science education

There are well founded claims to explicitly teach code reading. While students form some reading strategies on their own, reading skills benefit from explicit development efforts. They do not automatically improve together with advancing writing skills, as understanding existing code partly requires different activities than writing. A competent reader has several reading

strategies and can switch to one deemed suitable for the current task. Without a set of several strategies, a programmer can only resort to the few available ones, of which none might be a fitting choice. Besides, comprehension problems might at least partly result from reading problems. Teaching code reading can presumably attenuate some of such issues, especially when they are systematic. Reading programs, like writing programs, should be practiced throughout the curriculum, since its mastery takes time and effort [Busjahn & Schulte, 2013], [Busjahn et al., 2015a, 263,264], [Deimel Jr., 1985, 5-7], [Deimel & Naveda, 1990, 1,5,6,21,22], [Lister et al., 2004, 139], [Lister et al., 2009, 165], [Mannila, 2007], [Pea, 1986, 34], [Spinellis, 2003a, 1], [Xie et al., 2018], [Xie et al., 2019].

Skills that gained particular interest in computer science education are tracing code, explaining its purpose in plain words, and code writing. They are regarded as distinct skills, but feature certain dependencies [Xie et al., 2019, 205-208]. The development of these skills in novice programmers was specifically studied by the Leeds Working Group [Lister et al., 2004] and the BRACElet project [Clear et al., 2011]. Lister et al. [2004] call attention to the finding that the problem of unsuccessful students is not necessarily that they are weak at problem-solving, but rather missing precursor skills that relate more to code reading than writing. Some of the core findings of the BRACElet project were the relationships between performance on tracing, explaining, and writing code. The ability to trace code already correlates with the ability to write it, but the skills for code tracing and explaining combined are an even greater predictor for writing performance. Students who are weak at tracing code often also cannot explain or write it very well, on the other hand students who are reasonably good at code writing, usually posses the ability to trace and explain code. A hierarchy of programming related tasks is assumed with knowledge of basic programming constructs at the bottom, explaining code, solving Parson's problems, and tracing of iterative code as intermediate levels, and the ability to write non-trivial correct code at the top. However, it is not regarded as a strict hierarchy, in which the ability to trace precedes the ability to explain, and tracing and explaining precede the ability to write code. Rather, for most students a minimal level of tracing and explaining skills is needed for code writing, but by themselves are not enough to enable it [Clear et al., 2011, 3], [Lister et al., 2009], [Lopez et al., 2008], [Venables et al., 2009]. Xie et al. [2019] propose and evaluate a theory in which the four skills tracing, writing syntax, comprehending templates (reusable abstractions of programming knowledge), and writing code with templates are learned incrementally. The approach advocates teaching reading semantics before writing syntax, since it is the foundation to all other skills. It further differentiates between writing correct syntax and writing meaningful code with templates.

Busjahn & Schulte [2013] conducted interviews with programming instructors on the role of code reading and comprehension. The analysis shed light on different aspects of code reading in programming education. One striking point was that educators regarded reading as important, but seldom taught or used it directly. However, there are at least some studies and teaching approaches that focus on code reading. Based upon a survey on literature, existing programming courses and textbooks, Merrienboer & Krammer [1987] illustrate three instructional strategies for the design of introductory programming courses in high school and evaluate them. In the *Reading approach* students start by understanding non-trivial, well-designed programs. The tasks gradually become more complex, changing from using and analyzing, through modifying and extending, to designing and writing programs. In contrast, the *Expert* and the *Spiral approaches* put an emphasis on writing code. The first immediately covers complex, yet motivating problems with top-down design, while the latter is about starting with simple coding problems and basic language constructs, then gradually becomes more complex in small incremental steps. Merrienboer & Krammer [1987] identify six tactics for teaching programming, i.e. specific plans of action that are assumed to be beneficial, and assess how well they can be implemented in

each approach. Giving all tactics equal weight during the evaluation, the Reading approach was found to be superior to the Expert and Spiral approaches and allows to better control the processing load of students.

Selby [2011] also surveyed approaches to teaching introductory programming and extracts four common approaches. In *code analysis* students focus on reading and understanding existing source code, before they write their own. The *building blocks* approach corresponds to learning vocabulary, nouns and verbs, before constructing sentences. *Simple units* emphasizes mastering solutions to small problems before applying the learned logic to more complex problems. Finally, *full systems* resembles learning a foreign language by immersion. Learners work on non-trivial problems, and concepts and language constructs are only introduced when they are needed for the current step. Selby concludes that in order to teach programming effectively all of these approaches should be used in combination as programming competency requires mastering different skills and each approach allows the development of one or more of skills in a particular manner.

As for concrete courses, Kimura describes a reading-first introductory programming course as early as 1979 [Kimura, 1979]. The first half of the semester was spent exclusively on reading exercises, no algorithm design or presentation of the programming language was given. Students were expected to learn the language by carefully studying sample programs. Only the second half of the term was focused on writing own programs. The exam results seem to confirm that reading can precede writing and that reading skills can be acquired by working through example programs. Deimel Jr. [1985, 10,11] also outlines an introductory programming course that emphasizes the study of existing programs before writing. Eventually programs are modified and enhanced, and finally students mostly write their own programs. He compares it to learning a natural language, where writing skills follow upon reading skills. Campbell & Bolker [2002] teach introductory programming by immersion, similar to learning a foreign language. Students start by reading, modifying, and writing about an existing program. They deal with interfaces, architecture, and design questions early on, syntax is learned when needed. The instructors reason that modifying well-written code is easier than writing it, and that their approach allows to introduce real-world design issues early on. Under the rationale that students should first read and study software artifacts, before developing them themselves, Hilburn et al. [2011] present the inspection of software artifacts as active learning technique in software engineering education throughout the curriculum. Case studies are used to allow for real-world professional practices. Nelson et al. [2017] present a theory of program tracing knowledge together with a comprehension-first pedagogy, which teaches and assesses tracing skills without writing or editing source code. They introduce PLTutor, a tutorial system that allows to step through the control flow paths of example programs. Their evaluation study showed that participants working with PLTutor performed comparably or better than those using a sophisticated writing-oriented tutorial.

These approaches illustrate that code reading has its place in programming education and a number of advantages of a reading-centered approach to programming can be drawn. It allows the use of any programming language, pseudocode, or even structured English, and the code can be presented either on paper or in a development environment. Without having to master tools or environments, programming logic can be addressed early on. Furthermore, code reading facilitates the development of skills involved in debugging and helps to form the foundation for writing programs. However, some learners do not like to learn programming on paper, and not having immediate feedback as when programming on a computer makes this approach impractical for independent learning. Although not suitable in every situation, there are hints that a reading-oriented approach is especially appropriate for weaker students, as perpetual editing, compiling, and executing can be discouraging [Selby, 2011, 2,3]. Kimura [1979] states

the further benefits that students learn a given programming style before developing their own and recognize the importance of writing readable and understandable code. However, he also observed that students were partly frustrated with the reading-first approach, because they perceived learning from examples as inefficient. By reading code, students can learn about programming style, develop an intuition about what programs should look like, and therefore avoid some writing errors. Students can also work on programs that are yet too complex for them to write on their own and language features can be introduced in real-world contexts, which can raise students' motivation to learn [Campbell & Bolker, 2002, 23,24], [Deimel Jr., 1985, 10,11], [Hilburn et al., 2011], [Kölling & Rosenberg, 2001, 34]. Having received instruction on code reading should make students better code maintainers, as well as good learners even on the job, due to their greater ability to understand program examples [Deimel & Naveda, 1990, 6].

Deimel Jr. [1985] and Deimel & Naveda [1990] give ample suggestions on how to include code reading into programming education, even if the instruction is not reading-centered at all. Besides facilitating the learning of reading strategies, reading tasks broaden the spectrum of exercise and exam questions also with regard to code writing skills. Some advantages of reading tasks are that they are often easy to grade, quite objective, and a quite consistent measure of student performance. Both reports include instructions on how to construct good reading tasks, provide a framework for reading tasks based on Bloom's taxonomy of learning objectives, and present an extensive list of task and question types for assessing code comprehension, including sample exercises.

1.1.3 The nature of source code

> "[P]rograms have a dual nature - they can be *executed* for effect, and they can be *read* as communicative entities." [Soloway & Ehrlich, 1984, 595]

> "However, a program has two audiences:
> - *The computer*: The instructions in a program turn the computer into a *mechanism* that dictates *how* a problem can be solved.
> - *The human reader*: The programmer needs to have an *explanation* as to *why* the program solves the given problem." [Soloway, 1986, 850,851]

> "Source code is, among other things, a text to be read." [Raymond, 1991, 3]

> "In maintenance, the main role of source code is not as a compilable entity, but as a human-readable statement of the intent and mechanism of the program." [Raymond, 1991, 3]

> "Software source code is the definitive medium for communicating a program's operation and for storing knowledge in an executable form. You can compile source code into an executable program, you can read it to understand what a program does and how it works, and you can modify it to change the program's function." [Spinellis, 2003a, 1]

> "The original role of programming languages is that of a communication medium between a human and a computer. Today, the life span of software has increased, and programming teams have grown in size. As programmers need to communicate about software, computer code has also become an important *human* communication medium." [Dubochet, 2009, 174]

These quotes illustrate that source code is a very special type of text, as it is directed both at humans and computers, and it has the additional dimension of being executable. Text written in a programming language is markedly different from text in a natural language [Busjahn et al., 2015a, 255]. Source code actually contains words from natural languages, e.g. keywords and some identifiers. However, they occur in a very different setting [Blascheck & Sharif, 2019, 1]. While natural-language text is usually written in the order it is supposed to be read, in source code the order of statements is often not the same in which they are executed. Additionally, the order can change, e.g. when run with different parameters [Deimel & Naveda, 1990, 11], [Uwano et al., 2007, 2291]. Like natural-language text, source code can be divided into different types of elements on different levels of abstractions, e.g. keywords and identifiers, but also constructs like loops or functions. Source code is complex, highly abstract, and hardly contains redundancy, it also features a formally defined structure and layout. The level of complexity between different code areas varies substantially, e.g. between a simple assignment and a comparison with many elements or nested structures. The number of keywords, operators, and separators is limited, they are mostly comprised of only a few characters, and their semantic information is quite fixed, for identifiers and literals on the other hand there are almost indefinite options (see Binkley et al. [2012] for a detailed discussion on the impact of identifier names) [Busjahn et al., 2014a, 338], [Busjahn et al., 2015a, 255], [Crosby & Stelovsky, 1990, 24,25,34].

The special characteristic of source code that it can be read and executed is reflected in the Block Model, an educational model of program comprehension. The model applies the work of Kintsch [1998] on text comprehension, but also findings about program comprehension, learning programming, and difficulties of novices, as well as philosophical and psychological research on the characteristics of technical artifacts. It aims at providing vocabulary for learning and teaching processes in programming, to facilitate planning and analyzing lessons, as well as for research studies. As technical artifact, source code has a dual nature and comprises the aspects of *structure* and *function*. Structure is concerned with the empirically observable and objectively measurable properties of the program, thus its text surface and the resulting execution. Function refers to the goal or purpose of a program. Consequently, the Block Model distinguishes three dimensions of understanding a program text. The text surface is the external representation of the program from which information is extracted. It is the actual code a person reads. Program execution is the aspect which sets source code apart from other types of texts. Functions / goals of the program refer to the code's intention, the algorithmic idea. Execution has a crucial role for understanding source code and bridging execution and function is often quite a challenge. The three dimensions are structured as a matrix with several hierarchical levels of comprehension from single words (Atoms), to small units (Blocks), to inferences about the Relations between Blocks, to integrating the complete program (Macrolevel) [Schulte, 2007], [Schulte, 2008a], [Schulte, 2008b, 115,116], [Schulte et al., 2010, 69,70].
Consequently, source code is a very special type of text and reading it warrants closer inspection.

1.2 Research questions

Motivated by the importance of code reading for program comprehension, this work aims to deepen the knowledge about how programmers read code. At this, it concentrates on the measurable part of code reading - the visual behavior, since many cognitive processes are reflected in the eye movements. Two research questions (RQ) are devised to broaden insights on code reading and demonstrate the suitability of studying visual behavior during code reading.

As argued in 1.1.3 *The nature of source code*, source code is a special type of text and it is questionable that it is read just as natural-language text. While code reading shares some aspects with other forms of reading, several specifics emanate from the text itself [Busjahn & Schulte,

2013, 7,9], [Rooksby et al., 2006, 206], [Uwano et al., 2006, 134], [Uwano et al., 2007, 2291]. In their book on the "Psychology of Reading" Rayner et al. directly acknowledge that activities like reading code to find an error probably involves strategies and processes different from normal silent reading to understand a natural-language text [Rayner et al., 2012, 19]. Deimel & Naveda [1990, 11] doubt that reading code top-to-bottom is an adequate strategy, even if it might seem an obvious one at least to a novice programmer and claim that "[p]rograms are not read like novels" [Deimel & Naveda, 1990, 11]. Section 2.4 *Eye movements in programming* outlines a number of empirical studies that substantiate this notion. Drawing from previous work and the pilot study by Busjahn et al. [2011], two research questions are formulated.

Research questions:

1. Is reading behavior different between natural-language text and source code?
   If so, is the difference already present in novices?
2. Do novices exhibit a different code reading behavior than experts?

Stratifying participants into novices and experts allows to ascertain whether differences in reading behavior arise from the source code or are an effect of expertise. It also provides didactically useful data on whether experience is reflected in the visual behavior.

Many methodological challenges persist for eye movement studies. In order to address the research questions a number of technical and statistical approaches have been evaluated, adjusted, and developed. The resulting methodological framework is the main focus of this work. The contributions are advancing the research methodology in the domain of program comprehension and a deeper knowledge of processes involved in it. The suitability of the presented approach is demonstrated with the two research questions.

1.3 Structure of this work

The next chapter 2 *Background* discusses eye tracking and related work on eye movements in programming. Chapter 3 *EMCR study description* introduces the design of the study on eye movements in code reading (EMCR) including stimuli, data collection, and participants. Subsequently, chapter 4 *Detecting oculomotor events* describes how fixations and saccades were derived from raw gaze data. Chapters 5 *Eye tracking error* and 6 *Error correction* discuss errors in eye tracking data and how they are addressed before analysis. Chapter 7 details the *Analysis procedure*, chapter 8 the *Analysis measures*. Finally, the results are presented in chapter 9 *Analysis results*, followed by chapter 10 *Conclusion*.

2 Background

2.1 Why using gaze

A number of different methods have been applied to study program comprehension, e.g. think-aloud and its variants, interviews, questionnaires, cloze tests, and surveys [Bednarik & Tukiainen, 2006, 125,126], [Fan, 2010, 7,46-48,53], [Obaidellah et al., 2018, 1,2], [Sharafi et al., 2020, 3135], [Sharif & Shaffer, 2015, 807]. These techniques are only of limited suitability for the intended study on code reading. Some only record the outcome of the comprehension process after it was carried out. However, participants might forget parts of the process or some aspects might seem irrelevant after task completion and are thus not mentioned, e.g. an assumption about the code that was eventually dismissed during reading. Think-aloud is a valuable method to study comprehension, but it is hard to accomplish for the participants and affects their behavior. Verbalizing thoughts concurrently while carrying out a task imposes an additional cognitive load and the participant cannot fully concentrate on the task. When asked to think-aloud concurrently, participants often mainly give manipulative statements like "I click ...", but hardly refer to cognitive operations. Participants often have to be reminded to verbalize their thoughts, since it is a rather unnatural task and even expert programmers find it difficult to explain exactly how they read a program. Again, many aspects go unmentioned, since the person is either unaware of them or deems them unimportant. Furthermore, think-aloud like other self-reports is very subjective and descriptions of the same processes are highly variable [Busjahn et al., 2014c, 4], [Gog et al., 2009, 326], [Kerkau, 2011, 334-337], [Sharafi et al., 2020, 3135].

An instrument is needed that captures the undisturbed reading process. In order to study normal silent reading, participants have to carry out normal silent reading, not another task like naming isolated words or reading out loud, as these may significantly distort the process one wishes to study. The components of reading might not change radically from task to task, but that cannot be taken for granted [Rayner et al., 2012, 19,20]. Eye tracking is an established technique to study natural-language text reading and other information processing tasks like image perception and it is generally accepted that gaze and attention are strongly linked, especially in reading [Bente, 2004, 298,306-310], [Daw, 2012, 2,3], [Duchowski, 2017, 3,4,11-13,45], [Gog et al., 2009, 327-330], [Holmqvist et al., 2011, 378,379], [Liversedge et al., 2011, v], [Majaranta & Bulling, 2014, 39,40], [Sharafi et al., 2020, 3130,3133], [Rayner, 1998, 374,375,404], [Rayner et al., 2005, 97], [Rayner et al., 2012, 19,20]. Crosby & Stelovsky [1990, 28,29] affirm that gaze also indicates attention when reading source code. Sharafi et al. [2020, 3168] point out that eye tracking fits in well with software engineering research, since so many

of its activities involve visually-oriented artifacts. It provides information about what part of a code was of interest at exactly what point in time, thus achieving a fine granularity of data capture [Busjahn et al., 2014c, 4], [Schall & Romano Bergstrom, 2014, 3].

Aschwanden & Crosby [2006, 5] found no correlation between lines of code that study participants consider important for understanding and lines they looked at most and conclude that asking people about such aspects is not a reliable way to determine beacons or other important lines in code, since the accounts did not match the actual visual behavior. Eye tracking proved to be more objective. Uwano et al. [2007, 2290,2291] point out that evaluation attempts for code review methods using a number of different techniques have not been very successful, because the performance of the individual reviewer has a greater impact than the review method itself. Therefore, they suggest using gaze data to study the differences between good and bad reviewers more objectively and applying eye tracking identified a particular pattern called *scan*. The reviewer first reads the entire code briefly from top to bottom, before concentrating on details. Reviewers who took enough time for the scan performed better at finding defects, probably because the overall program structure was taken in and suspicious code was already spotted during the scan, while without it, problematic areas were missed. Sharif et al. [2012] replicated this experiment with a larger sample and came to similar results. Albert & Tedesco [2010] evaluated the reliability of self-reported awareness measures with eye tracking by asking participants if they noticed certain elements on a website. While self-reported awareness measures were found to be reliable in general, there was an error rate of 15% in their first experiment and 17% in the second, i.e. participants falsely reported noticing an element, or missed that they actually looked at it. These findings further strengthen the value of using eye tracking data.

An alternative to eye tracking which still allows to measure visual attention on a computer display is the restricted focus viewer (RFV). It blurs the screen except for a small area, which can be seen clearly, so that the general structure of the stimulus is visible, but cannot be perceived in detail. In order to look at another part of the screen unrestricted, the mouse has to be moved there. The RFV reflects to a certain degree human vision. A small region in focus can be perceived clearly, the surrounding area only roughly. The RFV is a cheap and non-intrusive technique, and participants can move freely while sitting in front of the screen. It works even for participants wearing glasses or eye make-up, and there is no data deterioration, e.g. due to blinks or lost signal, which can be issues when using eye tracking [Bednarik & Tukiainen, 2004a], [Bednarik & Tukiainen, 2004b], [Bednarik & Tukiainen, 2007], [Blackwell et al., 2000], [Jansen et al., 2003]. Blackwell et al. [2000] and Jansen et al. [2003] found that results obtained from an eye tracker and the RFV are in line with each other. Nonetheless, participants using the RFV took significantly longer to respond, probably due to the much slower arm and hand movements compared to the eye and because of the blurring it takes longer to move from one part of the stimulus to another. The RFV technique was applied in a programming context by Romero et al. [2002a], Romero et al. [2002b], and Romero et al. [2003] to study attention switching between multiple representations in a debugging environment. However, several issues were pointed out by Bednarik & Tukiainen [2004a], Bednarik & Tukiainen [2004b], and Bednarik & Tukiainen [2007] for using the RFV in such complex problem-solving tasks. They also studied attention switching between representations in debugging, but employed an eye tracker additionally to the RFV. In some regards, their results agreed with the previous findings. Debugging performance as well as the distribution of time spent on different areas are not influenced by the RFV, but it affected the visual behavior, especially of experienced programmers. The number of attention switches differed significantly between the blurred and unrestricted view. Furthermore, in the blurred condition the RFV recorded significantly less switches than the eye tracker. Participants recurrently looked at the blurred part of the screen without making the effort of moving the mouse there in order to see that partly clearly. Consequently, the RFV fails to

record such switches. Eye tracking data proved to be more objective, supporting the suitability of eye tracking for measuring visual attention. In addition, Bednarik & Tukiainen [2004a] and Bednarik & Tukiainen [2007] reported that experienced programmers tended to find the blurring bothersome, even though it did not hamper their debugging performance. Ultimately, the RFV did not prevail and has not been employed for visual attention tracking in programming after 2007 [Obaidellah et al., 2018, 25,26,33]. With regard to the two research questions, a fine level of detail is needed, so the focus window would have to be even smaller than employed by Bednarik & Tukiainen, and thereby very irritating for participants, resulting in a very unnatural task. Thus, while the RFV can be a valid technique in some contexts, for the intended analysis, eye tracking is a more suitable choice.

As a consequence, eye tracking is chosen as central instrument for empirical data collection, since vital information about cognitive processes can be inferred from a participant's gaze, which cannot be obtained from other methods. Besides, gaze data can complement and corroborate the results from other methods, resulting in a more thorough understanding of code comprehension. Eye tracking offers objective fine-grained data and affords less cognitive interference, participants are free to look at any part of the text as long as they wish and the technique has great ecological validity, since participants are engaged in undisturbed reading while the gaze is recorded [Bednarik & Tukiainen, 2006, 131], [Busjahn et al., 2014c, 3,4,9], [Gog et al., 2009], [Sharafi et al., 2020, 3128,3134,3135,3167,3168], [Sharif & Shaffer, 2015, 813,814]. Eye tracking studies on reading natural-language text as well as the findings from existing studies on code reading indicate that eye tracking is a promising methodological approach for studying cognitive processes during program comprehension (see 2.4 *Eye movements in programming*). Eye tracking also has its drawbacks, which will be addressed in detail in the chapters 5 *Eye tracking error* and 6 *Error correction*.
The following sections give an overview on gaze data and how it can be recorded.

2.2 The human eye and its movements

The eye is the sensory organ for visual perception. Unlike the ear, the eye is an active organ of perception and can be directed at a stimulus. Figure 2.1 provides a schematic representation of its anatomy. The eye ball has a spherical shape and is covered by three layers:

- Outer layer: The sclera keeps the eye in its shape and covers the majority of the eye ball. In the front, it passes into the transparent cornea, which allows light to enter the eye and is situated in front of the iris and pupil. Every blink spreads tear fluid over the cornea as protection against damage and infections.
- Middle layer: The iris regulates the amount of light that reaches the retina via the size of the pupil. The ciliary body has control over the shape of the lens and produces the aqueous fluid, which gives stability to the eye. The choroid contains blood vessels that supply oxygen and nutrients to the adjacent layers.
- Inner layer: The retina lines the rear inner surface of the eye ball and contains two types of photoreceptor cells that convert light into neuronal signals. Rods are very sensitive to even small amounts of light, but do not provide color vision. Cones on the other hand process color information and have a high spatial frequency. There are about 120 million rod and 6 million cone cells. Their density varies between different regions of the retina. If they were distributed evenly, the distance between cones would be too great to distinguish color information spatially in regular daylight. The highly pigmented area near the center of the retina, called macula lutea, contains the fovea centralis, a small pit which is densely packed with cones. This is the area of highest visual acuity in daylight. The rest of the retina is used for periphery vision, with lower acuity, but high sensitivity to light. No

photoreceptor cells are present at the optic disc where the optic nerve exits the eye ball resulting in a blind spot.

These ocular layers enclose the lens, the anterior and posterior chamber, which are are filled with aqueous fluid, as well as the vitreous body, a clear gel that fills the eye and helps to maintain its shape. Electromagnetic light waves enter the eye via the pupil. Then an inverted image is projected through the lens onto the retina, where the photoreceptors transform the light into electric impulses, which are sent to the brain via the optic nerve [Bente, 2004, 298,303,304], [Daw, 2012, 1-4], [Duchowski, 2017, 18-23], [Eysel, 2019, 723-743], [Holmqvist et al., 2011, 21,22], [Majaranta & Bulling, 2014, 40,41], [Rayner et al., 2012, 8-10], [Willoughby et al., 2010, 2-7].

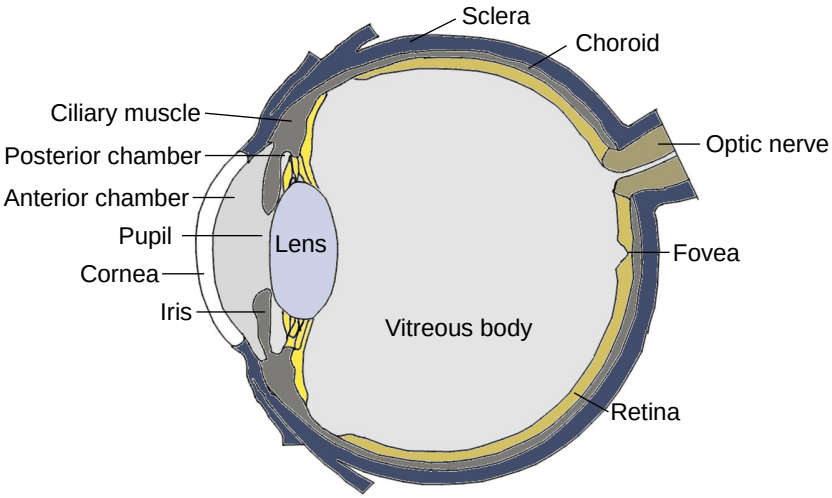


Figure 2.1: Anatomy of the human eye, based on Eysel [2019, 723]

Eye movements are complex processes and can only partly be controlled voluntarily. Mainly, the eyes make fast movements to bring an object in foveal vision, then stay fairly still for a short moment to take visual information in. Subsequently, they move to the next point. The brain integrates this piecemeal visual information into continuous images. The rapid movements, which bring a target into focus for detailed inspection, are called *saccades*. They occur in coordination so that the object is equally projected onto the fovea of both eyes. The duration of a saccade depends on the distance the eyes move, but typically lasts between 30 and 80 ms. Saccadic amplitude can reach more than 90° of visual angle, peak saccadic velocity up to 1000°/s, making saccades the fastest movement that humans are capable of. During saccades, visual intake is essentially suppressed as only blurry information would be perceived due to the fast movement [Bente, 2004, 303-305], [Daw, 2012, 3,4], [Duchowski, 2017, 4,15,17,39-43], [Eysel, 2019, 762-765], [Holmqvist et al., 2011, 21-23,312-315,321,322,326-329], [Majaranta & Bulling, 2014, 40,41], [Rayner, 1998, 373,378], [Rayner et al., 2005, 80-82].

Fixations are the events between two saccades when the eyes remain relatively stationary. Humans fixate about 10.000 times per hour. Very different values are suggested for minimal fixation duration. Holmqvist et al. [2011, 151,152,156,381] state that fixations can last as little as 30 or 40 ms, but sometimes minimal fixation duration is also set up to 250 ms. As for total fixation duration, very long fixations can take several seconds. Fixation duration is highly variable and influenced among other factors by the task at hand. For example, during silent reading mean fixation duration is 225 ms, while perceiving a scene they last on average 330 ms. In actuality, the eyes do not stay completely still during a fixation. Several intra-fixational movements of variable extent occur which occasionally make it difficult to identify fixations in raw gaze data, e.g. small tremors, drifts, or microsaccades. While fixation and saccades

are associated with attention, drifts and microsaccades are mostly physiologically determined [Bente, 2004, 304,305], [Blignaut & Beelders, 2009, 1-4], [Duchowski, 2017, 15,44,45], [Eysel, 2019, 763-765], [Holmqvist et al., 2011, 21-23,150,151,377-383], [Karsh & Breitenbach, 1983, 53], [Rayner, 1998, 373,374], [Rayner et al., 2005, 80-82]. The term *fixation* does not only refer to the period of relative physical stillness of the eye, but is partly also used for other related concepts, like the cognitive event of processing the fixated target. These concepts of fixation overlap to a great extent, but are not completely identical [Holmqvist et al., 2011, 150,151,377-380], [Nyström & Holmqvist, 2010, 197]. In this work, fixation denominates the oculomotor event that has been detected in raw gaze data during which the eye is relatively still, as is also common with others [Holmqvist et al., 2011, 150,151].

Other types of eye movements include pursuit, i.e. following a moving object with the eyes, vergence, when the eyes move into opposite directions, and vestibular eye movements to compensate for head or body movements [Bente, 2004, 305], [Duchowski, 2017, 39,43,44], [Eysel, 2019, 762-765], [Holmqvist et al., 2011, 23,24], [Majaranta & Bulling, 2014, 40,41], [Rayner, 1998, 373]. However, the focus of this work is on fixations and saccades.

Eye movements in silent reading are of special interest for the research questions.[1] In many European languages text is predominantly read line after line from top to bottom and from left to right along a line (see figure 2.2 and figure 2.3). At the end of a line a long leftwards saccade, called return sweep, brings the gaze to the beginning of the following line. Occasionally, the eyes move backwards in the text, such saccades are called regressions and are often associated with comprehension difficulties. Typically about 10 to 15% of the saccades during reading are regressive. Fixation duration, saccadic amplitude, and regression rate are influenced by the structure of the text, its formatting, as well as difficulty, e.g. when a text becomes more difficult, fixation duration and regression rate tend to increase, while saccadic amplitude decreases. Eye movements vary substantially between different readers, as well as for the same person within a single passage of text [Eysel, 2019, 763,765], [Holmqvist et al., 2011, 213,214], [Rayner, 1998, 375,376,387,392,393], [Rayner et al., 2005, 80-82], [Yamaya et al., 2017, 100].

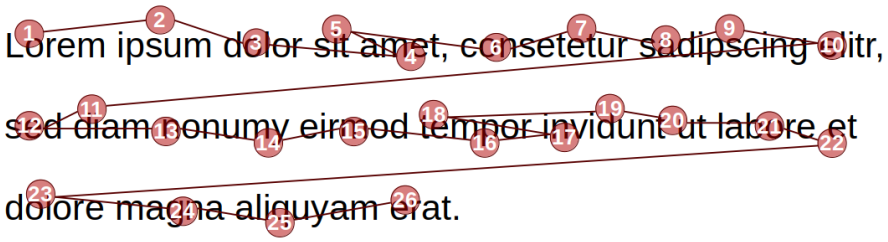


Figure 2.2: Prototypical forward directed reading pattern, including regressions and two return sweeps to the next line

Visual acuity is highest in the fovea, which spans about 1.5 - 2° of visual angle. In the parafoveal region, acuity is much lower and decreases even further in the periphery. Perceiving all visual information simultaneously with the same high acuity is too much to process, thus having central vision for detailed analysis and peripheral vision for noticing objects is more efficient. With regard to reading, the functional visual field is termed perceptual span. This span is asymmetrical and depends on reading direction. When reading alphabetic text left-to-right, it stretches about 3° of visual angle from the point of fixation to the right and hardly 1° to the left. The writing system also affects the overall size of the perceptual span, so do text difficulty and reading skill, e.g. the span is smaller for difficult text. Furthermore the visual field

[1] Other forms of reading, e.g. Braille text are out of scope of this work.

is larger horizontally than vertically [Daw, 2012, 2-4], [Duchowski, 2017, 15,29-33], [Holmqvist et al., 2011, 21,380,381], [Rayner, 1998, 374,378-381], [Rayner et al., 2005, 85-87].

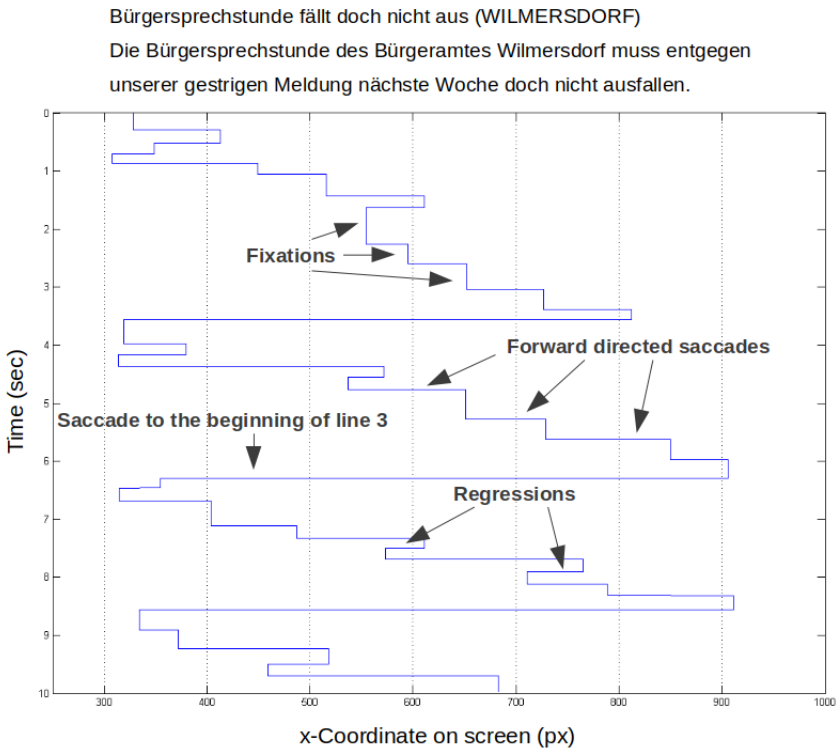


Figure 2.3: Eye movements during reading

2.3 Recording eye movements

In order to answer the research questions it is necessary to identify the point of regard (POR) on a computer screen, i.e. the screen location where the gaze is focused. For this purpose a video-based recording device with infrared pupil-corneal reflection tracking is used (see 3 *EMCR Study description*), which is currently the most prevalent method for capturing eye movements. With this type of eye tracking an infrared light source is directed at the eye, resulting in four reflections from different parts of the eye, the Purkinje reflections. Infrared light is used as it is invisible to the human eye and neither interferes with viewing nor influences pupil dilation. The first Purkinje reflection occurs from the cornea and is the brightest of the four. The POR is determined from the relative position of this corneal reflection to the center of the pupil. While it is possible to calculate the gaze location only from the pupil, having these two reference points is more robust and allows to track the gaze despite small movements of the head. Both, corneal reflection and pupil center, can be detected in video images of the eye (figure 2.4). When the eye moves, the relative position between them changes systematically and the gaze location can be calculated based on their relative distance. Therefore these eye trackers usually require some kind of calibration to establish the correspondence of corneal reflection - pupil relation to different gaze locations. Several factors, e.g. poor light conditions, may affect the recording and are discussed in detail in 5 *Eye tracking error*. Different types of infrared pupil-corneal reflection based eye trackers exist. Head-mounted devices are worn by the participant and mostly allow to capture a person's eye movements together with their environment. For the later presented study on code reading a more unobtrusive remote device is used, which is attached to the computer

display. There are a number of other techniques that measure eye movements, which are not suitable for the intended line of research. For example, electromagnetic coil systems do not provide the required POR measurements. Besides, this type of device is rather intrusive and uncomfortable to wear, since small coils have to be positioned onto the eyes. Electrooculography measures electric potential differences on the skin around the eyes. Again, measuring the POR is problematic and not accurate enough for studying reading [Bente, 2004, 310-317], [Duchowski, 2017, 49-56,59,85-87,121], [Holmqvist et al., 2011, 9,10,21,24-29], [Majaranta & Bulling, 2014, 40-47], [Schall & Romano Bergstrom, 2014], [Sharafi et al., 2020, 3130-3134]. A more detailed description of eye tracking systems can be found in Duchowski [2017, 49-198] and Schall & Romano Bergstrom [2014]. Wade & Tatler [2005, 1-32] provide an outline on the historical development of eye movement recording devices.

(a) Top left (b) Straight ahead (c) Bottom right

Figure 2.4: Video image captured by an eye tracker showing a participant looking at different parts of a computer screen. Corneal reflection and center of the pupil are marked with cross hairs. The images were provided by S. Tamm and are used with permission.

2.4 Eye movements in programming

Using eye movements to study programming was suggested as early as 1986 by Crosby [1986, 132]. The first actual study employing eye tracking in a programming context was published 1989 by Crosby & Stelovsky, investigating the interaction between presentation medium and programming expertise. A binary search algorithm was presented in Pascal and as slides from an animated graphic that demonstrated the algorithm. Several distinct viewing strategies emerged for both representations. Participants with low and high programming expertise partly used similar viewing strategies, thus these are not associated with a specific level of experience. Similarly, when analyzing the amount of time spent on comments versus code, code-oriented and comment-oriented participants were found in both expertise groups. Nevertheless, the low experience group spent on average significantly more time on comments than the high experience group. Crosby & Stelovsky [1990] add that experienced programmers focused more on complex code areas. Based on their findings they reason that when reading code, fixations can serve as indicator for attention. Furthermore, reading source code resulted in much more fixations as well as regressions than what is typically found for natural-language text, so it is concluded that algorithms induce different reading strategies.

Two related literature reviews were published in 2015 and 2018 and are briefly discussed here. Sharafi et al. [2015b] report on the usage of eye tracking and RFVs in software engineering, covering 36 publications from 1990 to the end of 2014. They note that eye tracking has increasingly been utilized in software engineering research from 2006 on. Five topic clusters emerged from their survey: most papers were published on code comprehension (12 papers), followed by model comprehension (10 papers), and debugging (9 papers). Collaborative interaction, e.g. during pair programming, was represented with three and traceability with two papers. The review also provides a comprehensive overview of the devices, stimulus materials, study participants, metrics, tools, and visualizations that were employed in the reported studies, as well as relevant findings and limitations. The metrics used were later refined in Sharafi et al. [2015a]. Sharafi

et al. [2015b] conclude that eye tracking is a beneficial tool for empirical studies in software engineering.

Obaidellah et al. [2018]'s review on eye tracking in programming includes 63 publications from 1990 to June 2017. This survey continues and validates the work by Sharafi et al. [2015b] with emphasis on programming and programming-related contexts. It compiles the published topics and study designs, summarizing stimuli, participants, devices (including RFV), metrics, and variables, as well as limitations. The survey yielded five topic areas compatible with the categories by Sharafi et al. [2015b]. Again program/code comprehension obtained the highest amount of publications (26 papers), followed by debugging with 19 papers. Program comprehension has continuously been studied from the beginning on and still retains an increasing interest. Likewise, eye tracking studies in debugging have grown since 2012. Non-code comprehension, which includes stimuli other than source code, e.g. UML diagrams, or focuses on other software engineering tasks, had ten papers, collaborative programming five, traceability three. The vast majority of the studies used source code or programming language stimuli, not diagrams or other materials.

Sharafi et al. [2020] complement these works with "A practical guide on conducting eye tracking studies in software engineering". They discuss why, when, and how eye tracking is appropriate, and compile use cases, metrics, visualizations, and statistical analyses for eye tracking data. As typical topics for eye tracking studies in software engineering, they identify program comprehension, diagram comprehension, code review, traceability, education, combining eye tracking with other psycho-physiological measures, and source code summarization. They classify research questions into the categories 1) evaluation of usefulness of systems, artifacts, or tools, 2) evaluation of the participants' effectiveness and efficiency when performing a specific task while using some systems, artifacts, or tools, 3) finding areas of interest in a stimulus by studying the distribution and the intensity of participants' visual attention, and 4) detecting navigation strategies used by participants by studying their scanpaths when performing software engineering tasks. Sharafi et al. [2020] emphasize that eye tracking contributes valuable insights, that cannot be obtained with other methods. Eye tracking has the potential to improve both the quality of artifacts in software engineering and the quality of the developers' interactions with these artifacts. However, it is cautioned that it also produces a lot of data that has to be carefully collected, stored, and analyzed to allow valid conclusions.

In the following, a number of publications will be discussed that are of particular interest with regard to the research questions or were not included in the reviews. Fan [2010] studied how beacons, comments, and task type influence program comprehension during software maintenance. Using gaze data it was found that the presence of comments and the type of task affect code scanning patterns as well as the ability of programmers to chunk code blocks. The presence of comments resulted in a more top-to-bottom, left-to-right reading order, while without comments participants' gaze moved around in the code to a greater extent. What is identified as beacon by a programmer is somewhat variable and depends among other factors on the type of task. Overall, gaze data was found to be helpful for documenting and analyzing program comprehension processes.

Nüssli [2011] focuses on remote collaborative problem solving, but as one aspect studied code reading from single and pair programmers. When reading individually, participants first looked at the major structural elements, presumably to grasp the general code structure. Then they performed a semantic code analysis by focusing their gaze on the computational parts of the code. Gaze patterns were found to be different when building a mental model then when updating it. Furthermore, how much the gaze is dispersed indicates to some degree processing depth. Episodes of low dispersion are associated with deep processing on complex specific parts, high dispersion with more general overview processing.

Hansen [2015] studied the cognitive complexity of source code. Programmers, ranging from novice to expert, were asked to predict the output of short Python programs, of which several different variations existed regarding spacing, naming, and function. For some of the participants gaze data was collected and helped to identify important code areas, characterize programmers' strategies, and investigate the source of participant errors. No significant correlation was found between eye tracking metrics and programming expertise or task performance, presumably because the task was rather unique. Fixation duration however did indicate the importance of code areas. Lines that were relevant to the task often drew the most attention. Keywords were the least frequently fixated tokens, while more complex statements also obtained more attention. Based on the errors, timings, and keystrokes recorded from programmers, two program comprehension models were developed. The first one, Mr. Bits, is built on the ACT-R cognitive architecture and predicts the eye movements and keystrokes of a programmer solving the experiment task correctly. The second, Nibbles, models the incorrect participants' interpretation errors.

Orlov [2016] studied the role of extrafoveal information processing during code comprehension and provides a gaze-contingent software for such studies. Even though study participants were able to solve the tasks when extrafoveal vision was restricted, the restriction affected both novice and expert programmers, though the effect was much stronger for experts. Without extrafoveal information, their behavior became similar to that of novices, as they made more mistakes and needed more time to solve the tasks than when unrestricted.

Binkley et al. [2012] investigated the impact of identifier style on source code comprehension for the two popular styles camel case and underscore using a combination of several data collection methods including eye tracking. The results of the different experiments are not fully congruent, but allow several general conclusions, e.g. expert programmers seem to be hardly affected by style. Regarding the central research questions of this work, it is of interest that several aspects from natural-language reading were different when reading identifiers and the authors conclude that reading natural-language text and source code are fundamentally different. As an example, when reading a natural-language text underscores provide better readability, for source code however it is camel casing. They advise caution about making assumptions on code reading based on natural-language comprehension studies and suggest that a lot of foundational research on code reading and comprehension is still needed.

Busjahn et al. [2011] and Busjahn et al. [2014a] illustrate that eye tracking as well as associated measures and analyses from natural-language text reading can successfully be applied to code reading, however the actual findings are only transferable to a limited extent. Busjahn et al. [2011] conducted a study on the differences between reading natural-language text and source code. Even though there was considerable variability between participants as well as between texts, the mean fixation time found for source code was considerably longer than for natural-language text. Furthermore, when reading source code, participants made significantly more backwards movements in the text than for natural-language text, indicating that source code brings about different reading patterns than natural-language text. Busjahn et al. [2014a] analyze the attention distribution on source code elements in Java. When reading natural-language text, the time a reader looks at a word is influenced by a number of factors, like word length, word frequency, and grammatical category. The study results show that for source code the type of lexical element also influences visual attention, however only for one type. Separators got substantially less attention than identifiers, operators, keywords and literals. In natural-language text reading, less frequent words tend to induce longer fixation durations and dwell times. However, at least for keywords element frequency proved not to be a relevant factor in the variability of first fixation duration and first dwell time. Applying measures and analyses from natural-language text reading to source code is a valuable direction of study, however not

without careful review. Peterson et al. [2019] draw on this study to examine factors that influence dwell time during source code reading, using large open source Java projects as stimuli and taking the analysis from token-level to lines. Participants were novice and expert programmers who were asked to summarize methods. For analysis, they distinguish nine semantic types of lines: Method Call, If, Variable, Method Signature, For, Import, Class Attribute, and Class. As in the earlier study, the dwell time distribution exhibits a right-skew and was therefore log-transformed, resulting in a similar distribution to Busjahn et al. [2014a]. While elements with more characters take longer to read, no such trend was found between length of the lines and total dwell times. Also, no correlation was found between line frequency and duration of the first fixation, corroborating this finding from Busjahn et al. [2014a] for keywords. Peterson et al. [2019] further analyze the distribution of visual attention over the line types with regard to first and last fixation, differences in distributions between small and large methods, and expertise. The line types Method Call, If, and Variable received the most visual attention in terms of dwell time. The first fixation in a method most frequently lands on a line of the types Method Signature, Method Call, or Variable. While participants on average spent more time in large methods, the average time per line is significantly higher in shorter methods. No significant differences were found between the dwell time distributions of novices and experts over the different line types. All in all, this work further adds to the findings about attention distribution on source code, bringing out specifics on line-level and emphasizing that reading source code is different from reading natural-language text.

Furthermore a workshop series emerged in 2013 which focuses on eye movements in programming and resulted in a number of publications, among others the workshop reports and proceedings [Bednarik et al., 2014], [Busjahn et al., 2015b], [Bednarik et al., 2016], [Tamm et al., 2017], [Bednarik & Schulte, 2018], [Begel & Siegmund, 2019], [Siegmund et al., 2019]. A publication that ensued from the first of these workshops explicitly introduces eye tracking as an instrument for computer science education research, discusses data analysis methods and challenges, together with tools to address them [Busjahn et al., 2014c]. They found that eye tracking in computing education is feasible, yields rich data for analysis, and brings about novel teaching ideas. To demonstrate possible outcomes of employing eye tracking, a tiered coding scheme for gaze data is presented that captures objective visual behavior as well as the coder's inferences about the programmer's comprehension. Additionally they offer a broad selection of topics and questions from computing education that can be addressed with eye tracking, like better understanding the learner and comprehension challenges, as well as advancing instruction materials and tools. Busjahn et al. [2014b] further discuss options to combine quantitative and qualitative methods to develop coding schemes for gaze data in program comprehension studies, suggesting that automatically generated and assigned codes can be coupled with qualitatively derived ones in order to capture aspects of program comprehension.

Busjahn et al. [2015a] compared the eye movements of novice and expert programmers reading natural-language text and Java programs. They describe and validate several metrics capturing linearity in reading, i.e. how closely readers follow the top-to-bottom, left-to-right order in which words are written in Latin script languages. While there are differences in how novices read natural-language text and source code on a local level, the general reading approach is comparably linear on both. They exhibit clear linear trends on source code, even though they read it less linearly than natural language text. The non-linear portion might be caused by novices searching around for comprehension cues. Novices also skip more words in source code reading than on natural-language text. Experts exhibit significantly different reading behaviors than novices. They read code less linearly and skip more parts of the code. It was also found that the order in which experts read code is closer to the order in which the code is executed than how it is written down, suggesting they trace at least parts of the code. These findings suggest

that non-linear reading skills increase with expertise. This work sparked a number of replication studies. Peachock et al. [2017] conducted a replication, using C++ instead of Java. Both novice and non-novice participants read natural-language text more linearly than source code, corroborating that source code motivates different reading patterns than natural-language text. No major differences were found between novice and non-novice programmers. The authors surmise that this is due to the fact that the non-novices were still students, not experts as in the earlier study and therefore might have rather similar reading skills as the novices. Spinelli et al. [2018] studied visual attention patterns for dynamic code presentation techniques. To evaluate several types of code animation, they examined the participants' reading approach, operationalized using most of the measures presented by Busjahn et al. [2015a]. The observed reading approaches were in line with logical explanations for animations (i.e. increased top-to-bottom line reading when viewing a linear animation of line typing). They also found that viewing animations of a single textual representation of source code may affect the attentional processes of novice programmers during subsequent tasks and identified a possible effect on participants' attentional processes and element coverage when presenting live-written code. Blascheck & Sharif [2019] also draw on the study by Busjahn et al. [2015a] to compare natural-language and code reading. Their participants were novices and non-novices and C++ was used as programming language instead of Java. Additionally to using the quantitative linearity metrics, gaze data was analyzed qualitatively with radial transition graph visualizations, a donut chart, which depicts areas of interest (AOI) as segments and transitions between AOIs as arcs, whose thickness indicates the transition count. Natural-language text had a higher AOI coverage than source code, where novices tended look at more AOIs than non-novices. Overall, participants focused more on areas containing core functionality and skipped constructs not necessarily needed to answer the task correctly, e.g. brackets. Natural-language text was mostly read linearly, especially at the beginning. After having read the text completely, participants partly re-read or skimmed the text in a less linear order. On source code stimuli, both novices and non-novices had much more backward-directed transitions than on natural-language text. However, patterns representing linear as well as execution order were found. Several participants also carried out a linear first reading of the code. This study is overall in line with the previous findings and the additional analysis method provides deeper information on how the programmers distribute their visual attention.

Eye tracking can be used to address a multitude of topics in software engineering, benefiting practitioners and advancing computing education [Aschwanden & Crosby, 2006, 6], [Busjahn & Schulte, 2013, 3,4,9,10], [Busjahn et al., 2014c, 3,8,9], [Busjahn et al., 2015a, 263,264], [Deimel Jr., 1985, 5], [Deimel & Naveda, 1990, 1,5,6], [Rooksby et al., 2006, 210], [Schulte, 2007, 307], [Sharif & Shaffer, 2015], [Spinellis, 2003a, xxi,1]. Sharif & Shaffer [2015] describe two main uses for eye tracking in software development: assessing artifacts, tools, and techniques as well as using gaze to improve tools and tasks such as providing developer recommendations and software traceability links based on what the developer looks at. Information gathered by eye trackers can thus support common software development tasks and reduce developers' effort, making the work more developer-centric. Gaze data can further help to devise programming languages, that better satisfy programmers' needs, and development environments, that increase programmer's efficiency and code quality [Aschwanden & Crosby, 2006, 6]. Busjahn et al. [2014c, 8,9] list a number of subjects from a computer science education perspective, e.g. studying the effects of different representations and programming paradigms, the behaviors and strategies of learners, identifying their challenges, evaluating visualization tools and integrated development environments (IDE), and developing new tools to assess learners. Knowledge about experts' strategies can be used to develop respective teaching materials. After they have been taught reading techniques, eye tracking can be used to assess how well students are using them, thus

allowing a very fitting evaluation of such interventions. Integrated into an IDE, eye tracking can provide feedback and targeted assistance to the student, the instructor, or a professional programmer, raise awareness of their code reading behavior, and facilitate meta-cognition, e.g. informing the programmer that the initial scan of the program was not sufficient or that someone trying to find a bug is concentrating on completely unrelated code. Eye tracking is also a valuable instrument for other areas of computing education, e.g. understanding graphical data models [Busjahn et al., 2014c, 3], [Busjahn et al., 2015a, 263], [Sharif & Shaffer, 2015, 813].

These studies and groundwork demonstrate that the topics of code reading behavior and how it is different from natural-language reading is a valuable line of research, yet they substantiate the necessity of further research and many methodological challenges remain. The gathered experiences with data collection, cleaning, and analysis serve as basis for designing, conducting, and analyzing the further empirical work.

3 EMCR study description

3.1 Synopsis

Two eye tracking experiments were devised and conducted in order to collect eye movement data from novice and expert programmers. The designs of both experiments include first reading natural-language text, then source code. Due to their representativeness and wide use, English was chosen as natural-language and Java as programming language, partly subjoined with pseudocode. The same English texts were used for both experiments. Furthermore, two programs were presented to both novices and experts to allow for comparisons according to level of expertise. The novices were recruited from an introductory Java course for non-computer science students. The design of the novice study stipulates multiple recordings of eye movements during code reading at different stages of the learning process. For the expert study, professional programmers with diverse work backgrounds were recorded in per participant sessions, which also encompassed an interview about code reading.

3.2 Study design

Participants were asked to read and understand a number of stimulus texts while their gaze is recorded. After each text, they had to answer a comprehension question. The stimulus texts were designed to allow as much generalization as possible. The materials consist of three natural-language texts (NT), which were shown to both novice and expert programmers, as well as six different source codes (SC), of which two were read by both novices and experts. The NT reading was included in order to collect basal information about the participants' customary reading behavior and to obtain a baseline for comparison with code reading. It is to be assumed that SC at least partly elicits a different reading behavior than NT, thus it is of interest to which extent measures and findings from NT reading can be transferred to code reading.

Three task types were chosen to reflect several central tasks in program comprehension. The same task types were employed for NT as for SC stimuli in order to have comparable designs. For the *summary* task, participants were asked to write a summary of the text. On SC stimuli, the instruction for expert programmers included the additional prompt to describe the algorithmic idea and its implementation. This was not part of the instructions for novices, since it cannot be ensured that they understand the terms. The second task type is a *multiple-choice* question on the content of the text. The third type is an *outcome* question. On NT, participants were asked about a detail of the text, on SC they had to provide the value of a

certain variable after the program was executed. Tracing the value of a variable is a typical task during programming, especially when debugging. Being able to emulate program execution is a distinct skill, which can be regarded as fundamental for more complex programming activities like writing and maintaining code. It also represents a long-standing and often used task in programming education [Deimel Jr., 1985, 13], [Lister et al., 2004], [Lister et al., 2009], [Lopez et al., 2008], [Nelson et al., 2017], [Perkins & Martin, 1986], [Soloway, 1986, 857,858], [Venables et al., 2009], [Xie et al., 2018], [Xie et al., 2019]. Participants were informed about which kind of task they will be given before the stimulus was shown. They were also made aware of the option to choose "I don't know", in case they are not able to provide an answer. To create a balanced design and avoid effects from the order in which texts are shown, the presentation order was randomized, and so was the combination of stimulus texts and tasks, whereat the task types were assigned evenly. Figures 3.3 and 3.5 show exemplary texts and tasks for NT and SC respectively, the appendix contains all stimuli and tasks (A.2 *Natural-language stimuli*, A.3 *Source code stimuli*), figure 3.6 specifies how order and task type were arranged over the recordings.

The stimulus materials were presented in English, but participants could choose between answering the tasks in English or German. To make the text well readable, suggestions for screen text were followed [Kerkau, 2011, 332]. The font size varied between 13 and 16 pt, spacing between 1.5 and 2. The different sizes are due to the different lengths of the texts. To facilitate the later mapping of fixations to words, always the largest reasonable option was chosen, which allowed to fit the text onto the screen without scrolling. Longer codes were therefore mostly presented in font size 13 or 14 pt and 1.5 spacing, short codes in font size 16 pt and with a spacing of 2.

The procedure of the recordings was always the same. The first slide informed participants about the topic of the experiment and what stimuli to expect (see figure 3.1 for an example). Then the instruction for the first stimulus was shown (figure 3.2), followed by the stimulus text. When the participant was done reading the stimulus, the comprehension task was presented. In order to move from one slide to the next, participants had to actively press a button, when they were ready to proceed. The transition between slides was not automated, thus they could remain on any slide as long as they chose. There was no possibility to go back to a previous slide, so after getting to the slide containing the comprehension task, the question had to be answered without looking at the text. If participants were allowed to switch between stimulus text and comprehension task, some might not have tried to really understand the text, but rather used the task slide to look for clues and go back and forth between text and answer. Especially for multiple-choice questions, it can be tempting to have a look at the given options and then just scan the text for the right answer. However, the intention is to record reading behavior during comprehension. Alternating between text and answer slide would interrupt the process, and the captured data represented verification of given answers instead of comprehension.

The stimulus texts were presented on a computer screen. Gaze data was recorded with a RED-m eye tracker from SensoMotoric Instruments, which is aimed at use in scientific research. It utilizes the corneal reflex and is supposed to work robustly for a wide range of recording situations and participants, including those with visual aids. According to the manufacturer, this eye tracker has a spatial resolution of 0.1° (root mean square), an accuracy of 0.5°, and two sampling rates: 60 and 120 Hz. Furthermore, it compensates head movements to a certain degree [SensoMotoric Instruments, 2014], [SensoMotoric Instruments, 2016a], see also Mele & Federici [2012]. The RED-m is a small remote device, which is fastened to the frame underneath the computer display, so participants are not attached to any equipment or restricted in any way. Since some head movements are allowed, participants were not restrained with a head- or chin-rest, as that would make the situation very artificial. However, they were asked to lean

> This experiment is about program comprehension. There will be three source codes together with different comprehension questions. Please don't guess the answers. The codes do not contain bugs.

Figure 3.1: Sample of the first slide shown before every SC recording

> Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked to give a SUMMARY of the code.

Figure 3.2: Sample instruction

against the backrest of the chair and find a comfortable sitting position before the recording to reduce movements without causing strain for the participants. Overall, this setup makes the gaze recording rather unobtrusive. Gaze data was recorded with the OpenGazeAndMouseAnalyzer[1] (OGAMA), an open source software for recording and analyzing eye and mouse movements, which allows to integrate a number of different eye trackers [Voßkühler et al., 2008]. At the beginning of each recording, a calibration was carried out, followed by a validation. If necessary, this procedure was repeated until the calibration was satisfactory. During the recordings, the eye tracker was frequently recalibrated to maintain good data quality. With a few exceptions the gaze was recorded with a sampling rate of 120 Hz.

Participants knew that their gaze is being recorded and were made aware that the recording is not an exam and that they cannot do anything wrong. Furthermore, the researcher conducting the recording did not look over the participants' shoulders, but gave them space to work on the tasks without being under constant observation to reduce the effect of participants adapting their behavior because they feel watched [Sharafi et al., 2015b, 100], [Sharafi et al., 2020, 3155].

The design of a study strongly influences the results, e.g. a different set of stimuli may induce other findings. Such concerns are addressed in detail in chapter 9.5 *Threats to validity*. The EMCR study was designed very carefully, taking common pitfalls of eye tracking studies, like the eye tracker's limited accuracy, into account.

3.2.1 NT reading

In order to obtain the participants' customary reading behavior and allow comparisons between NT and SC reading, the EMCR study includes NT stimuli. English was chosen as natural-language, since it is the most spoken language worldwide and third with regard to native speakers [Ethnologue, 2020], [WorldAtlas, 2019]. Furthermore it is often used in reading research involving eye movement data, e.g. Rayner [1998], Rayner et al. [2005]. It can also be regarded as exemplary for a number of other languages that are read from top to bottom and left to right. Three short English texts were taken from an English language test[2]. They contain a mixture of frequent as well as rare words and the lengths of the words vary substantially. The short length of the texts allows to display them on the screen without scrolling. The types of comprehension questions for

[1] http://www.ogama.net, last accessed 12/05/2020

[2] NT1: https://www.ielts.org/pdf/115015_academic_reading_sample_task_-_matching_features___2_.pdf,
NT2: https://www.ielts.org/pdf/academic reading sample task diagram label completion.pdf,
NT3: https://www.ielts.org/pdf/115016_academic_reading_sample_task_-_matching_headings___2_.pdf,
last accessed on 06/08/2015

the NT stimuli are the same as for the SCs. Participants were either asked to write a summary of the text, answer a multiple-choice question about its content, or provide a specific detail from the text. The comprehension tasks on NT serve as motivation for the participants to read the texts thoroughly, and allow familiarization with the experimental procedure and task types, since the NT recording was always carried out first. The three NT stimuli are referred to as NT1, NT2, and NT3. Figure 3.3 provides NT1 together with its possible tasks. All NT stimuli can be found in the appendix (A.2 *Natural-language stimuli*).

The invention of rockets is linked inextricably with the invention of 'black powder'. Most historians of technology credit the Chinese with its discovery. They base their belief on studies of Chinese writings or on the notebooks of early Europeans who settled in or made long visits to China to study its history and civilisation. It is probable that, some time in the tenth century, black powder was first compounded from its basic ingredients of saltpetre, charcoal and sulphur.

Comprehension questions

Multiple-choice:
Which statement describes the content of the text?

- Black powder was probably discovered some time in the tenth century by the Chinese. It is essential for the invention of rockets.
- Historians believe that black powder was discovered by the Europeans after visits to China.
- Based on Chinese and European accounts, rockets were essential for the invention of black powder.
- I'm not sure.

Outcome:
Where was black powder supposedly discovered?

Summary:
Please give a summary of the text.

Figure 3.3: NT1 with comprehension tasks

3.2.2 SC reading

Java was chosen as programming language for the SC stimuli, since it is widely used both in industry and in programming education [Guo, 2014], [Mason et al., 2012], [Mason & Cooper, 2014], [Murphy et al., 2017]. In the TIOBE Programming Community Index[3], which provides a monthly information on the popularity of programming languages, Java was on place one at the time when the experiment was designed and remained among the top two ever since. Besides, from time to time it is by far the most popular programming language (figure 3.4). Java is object-oriented, but also contains many elements common in imperative and procedural programming, making it at least partially representative for a number of other languages. Java

[3]https://www.tiobe.com/, last accessed 01/15/2020

is also often used in other eye tracking studies [Obaidellah et al., 2018, 15], [Sharafi et al., 2015b, 90,92], thus facilitating comparisons with other work.

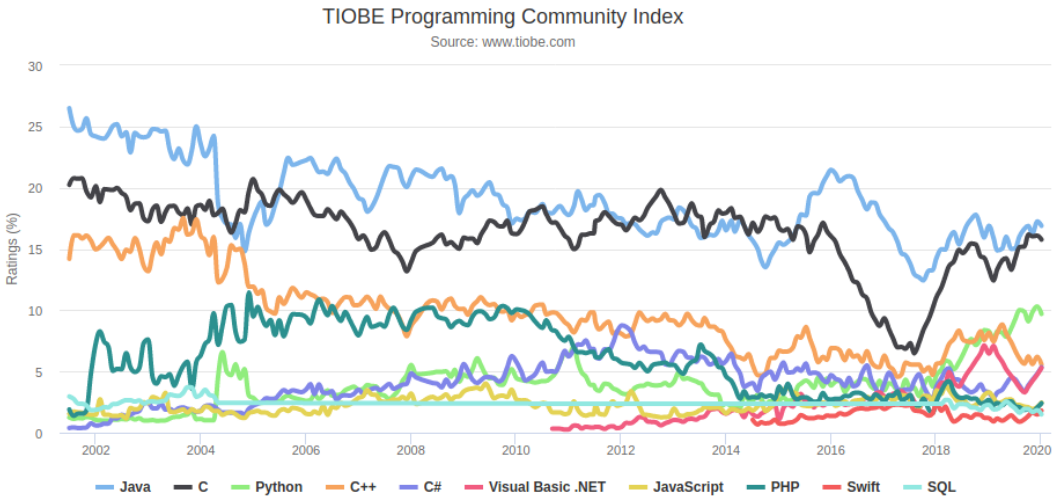


Figure 3.4: TIOBE Index, source: https://www.tiobe.com, status of 01/15/2020

The complexity and make of previous stimulus material in eye tracking studies on program comprehension were taken into account when composing the EMCR stimuli, e.g. Aschwanden & Crosby [2006], Bednarik & Tukiainen [2006], Hansen [2015], Uwano et al. [2006]. Furthermore, the stimuli cover fundamental concepts like loops and conditions. To allow to directly compare the code reading behavior of novices and experts, two programs were shown to both groups of expertise. These programs have to be easy enough, so novices can understand them after a few lessons, yet not too simple for the experts. All stimulus programs achieve something sensible in themselves and compile without errors. Participants were informed about the latter prior to stimulus presentation. In order to generate valid stimuli, they were mostly devised from teaching material and stimuli from other studies on program comprehension to allow for comparisons. Additionally, the programs largely use rather descriptive and meaningful names for variables and methods. To facilitate the mapping of gaze to single code elements for the analysis, a space character was added to the source code, wherever the syntactical rules allowed it, for example `String [ ]` instead of `String[]`. Furthermore, the stimulus programs do not contain syntax highlighting, because the participants were accustomed to different color schemes. The novices were at least acquainted with BlueJ[4], an IDE especially for Java, and the online coding environment provided by Udacity (see 3.2.2.1 *Novice programmers*). The expert programmers most likely use more professional IDEs and varying color schemes. If someone is used to a certain type of syntax highlighting, it will somewhat guide their visual attention. If there was highlighting, even a widely used one, like the default from Eclipse[5], it would be familiar to some, but not all participants. To create the same conditions for all participants and keep formatting consistent between the two groups, the stimulus programs were formatted plainly without color. Using C#.Net as programming language, Beelders & Plessis [2016a] and Beelders & Plessis [2016b] studied whether presenting code with and without syntax highlighting to IT students influences the reading behavior. Their participants indicated that they subjectively prefer the version with syntax highlighting, but no significant effect was found for any of the analyzed measures (number of fixations, fixation durations, and number of regressions). Thus, the absence of syntax highlighting should not have a measurable impact on code reading behavior.

[4]https://www.bluej.org/, last accessed 12/05/2020
[5]https://www.eclipse.org/ide/, last accessed 12/05/2020

3.2.2.1 Novice programmers

In order to recruit novice participants, a free weekly Java introduction was offered in form of an open continuous course at Freie Universität Berlin. It was directed at all interested persons not enrolled in a computer science program or related subject. The course was advertised via bulletin boards at the university and a fellow lecturer called the attention of his students to it. Students could gain credit points for their participation, but no grade was given. Participants filled in a questionnaire, which collected basic demographic data and further vital information on aspects like English proficiency, programming experience, and visual aids. Furthermore, they gave written consent to participate in the study. The questionnaire can be found in the appendix (A.1.1 *Questionnaires - Novices*).

Participants individually worked through an online course provided by Udacity[6]. However, a tutor was present to answer questions and help when students got stuck. Furthermore, students were encouraged to seek help from their classmates. An existing set of materials was chosen in order to expose all students to the same instruction. Furthermore, this ensures that the instructor, who conducted the course, does not impose any assumptions about reading behavior on the participants and therefore influences their behavior. The course covered six lessons, which were supposed to be worked through over a period of about three months. Due to the open nature of the course only few participants completed the course within this time frame. Participants occasionally took breaks between course sessions because of exams or work obligations.

The first recording consisted of the three English texts. After participants finished the first lesson of the Java course, their gaze was recorded while they read three SC stimuli and answered the respective comprehension questions. This exercise was repeated after each lesson, but only two other recordings are of interest. After lessons 3 and 5, programs were shown that are also part of the expert study. The recording setup was the same for all lessons, with outcome, summary, and multiple-choice questions. In total, there are five stimulus SCs for novices, three programs in lesson 1 (L1_SC1 - L1_SC3), one in lesson 3 (L3_SC1) and another in lesson 5 (L5_SC3). The identifier of the stimulus refers to the lesson and program number within that lesson. The stimulus programs were adjusted to the content of the respective lesson and included the main concepts that were introduced. The first three stimulus programs were written to reflect the concepts presented in lesson 1, one simple Java code for printing the result of an addition, and two pseudocodes with loops, which had been introduced without the respective Java syntax. Program L3_SC1 includes a class `Rectangle` and methods to calculate its width, height, and area. It originates from the eyeCode study on Python programming [Hansen, 2015, 39,40,101-105,206-208] and was transferred to Java. The program L5_SC3 is based on a classroom example provided by a German high school computer science teacher and was slightly revised, e.g. the German variable names were switched to English and it was shortened to fit a screen page. It defines a `Vehicle` that can change its speed. This program only contains programming constructs the novices are familiar with, yet it is rather difficult to grasp fully, since the `accelerate`-method allows not just to increase the speed, but also to decrease it via negative values. This difficulty was included to make the stimulus text simple enough to be understood by novices, but still sufficiently challenging for advanced programmers, as this program is also to be shown to the expert participants.

To keep the procedure consistent between recordings, the eye tracking sessions after lessons 3 and 5 each also contained three SCs, even though only one is of interest for analysis. This allows to randomize the order in which the stimuli are presented instead of only showing the target program right away. Besides, the programs and accompanying tasks also served as comprehension evaluation within the Java course and a single SC would have been meager for that purpose.

[6]https://www.udacity.com/course/cs046, last accessed 12/05/2020

Figure 3.5 shows an exemplary stimulus program together with its comprehension questions. Table 3.1 lists all source codes with their lines of code. The complete set of SC stimuli for novices can be found in the appendix (A.3.1 *Source code stimuli - Novices*), an overview of the recordings is provided in figure 3.6.

```
public class PrinterClass {
   public static void main ( String [ ] args ) {
     System.out.print ( "answer=" ) ;
     System.out.println ( 40 + 2 ) ;
   }
}
```

Comprehension questions

Multiple-choice:

What is the output?

- Answer=42
- answer=
  42
- answer
  =
  42
- I'm not sure.

Outcome:

What is the program's output?

Summary:

Please give a summary of the program.

Figure 3.5: Program L1_SC1 with comprehension tasks

The recording situation was designed to be very ecologically valid. Almost all recordings were carried out in the same computer class room in which the course took place, not a special eye tracking lab, at the same place and under the use of the same display. In this class room, it was not possible to completely shut out daylight, so the light conditions were not ideal, but recording there allowed to integrate the gaze recording smoothly into the course and made the situation more natural for the participants, since they were familiar with the environment. Only for one novice, the NTs and lesson 1 were recorded in another lab room, which likewise did not allow to fully control ambient light. Overall, the setup was aimed at creating an unobtrusive and natural situation.

3.2.2.2 Expert programmers

A number of professional programmers were recruited via personal contacts. All participants programmed professionally at the time of the recording, i.e. they were paid to program. As an incentive, they were given a goody bag with sweets. First, participants filled in a similar

questionnaire as the novices, which assessed demographic information, English level, programming experience, and visual characteristics. They also gave written consent to participate in the study (see A.1.2 *Questionnaires - Experts*).

Thereafter, the experts read the same three English texts as the novices while their gaze was recorded to gain baseline data about their reading behavior and familiarize them with the recording procedure. For experts three programs are of interest. SC1 is identical to the `Rectangle`-program L3_SC1. SC2 is also adapted from the eyeCode study [Hansen, 2015, 31-33,83-87,197,198] and presents the class `Quantities`, which finds the common elements of two lists. SC3 is the same as the `Vehicle`-program L5_SC3. Again, the three types of comprehension questions outcome, multiple-choice, and summary were used. For the summary task, experts were additionally prompted to tell what the algorithmic idea is and how it is implemented. Since novices are probably not familiar with those terms, they were only asked to summarize the program. The SC stimuli are provided in the appendix (A.3.2 *Source code stimuli - Experts*), an overview of the recordings is shown in figure 3.6. Table 3.1 provides the lines of code for each stimulus program. The study was conducted in Germany and Finland and attracted programmers with various backgrounds and different nationalities. Experts were mostly recorded at their workplace using their own display and keyboard, so they were also very familiar with the environment and the recording took place in a naturalistic situation. Just as with the novices, the researcher left the participant to the task during the recording and did not hover about.

After the eye tracking part of the study, a short guided interview was conducted in order to collect additional information about the experts' code reading behavior. First, participants were asked about their approach to read and understand the programs. Then, it was of interest whether they adapted their approach according to the task at hand. Finally, they were asked, if they think the approach(es) they described would be suitable for novice programmers as well. However, the questions were slightly adapted during the interviews to fit the participant's reactions. The interviews were recorded and transcribed (see A.4 *Expert interviews*). Two interviews were not used due to insufficient sound quality. The obtained responses mainly serve as background for informing measures to capture visual behavior during code reading.

Interview guideline:

- What is your approach to read and understand source code?
- You had to solve different kinds of tasks. Did that change your approach?
- Do you think this approach would be suitable for a novice programmer as well?

Stimulus text	Lines of code	Programming language	Description
L1_SC1	6	Java	Novices after lesson 1
L1_SC2	6	Pseudocode	Novices after lesson 1
L1_SC3	4	Pseudocode	Novices after lesson 1
L3_SC1	18	Java	Novices after lesson 3
L5_SC3	22	Java	Novices after lesson 5
SC1	18	Java	Experts, identical to L3_SC1
SC2	22	Java	Experts only
SC3	22	Java	Experts, identical to L5_SC3

Table 3.1: Stimulus source codes

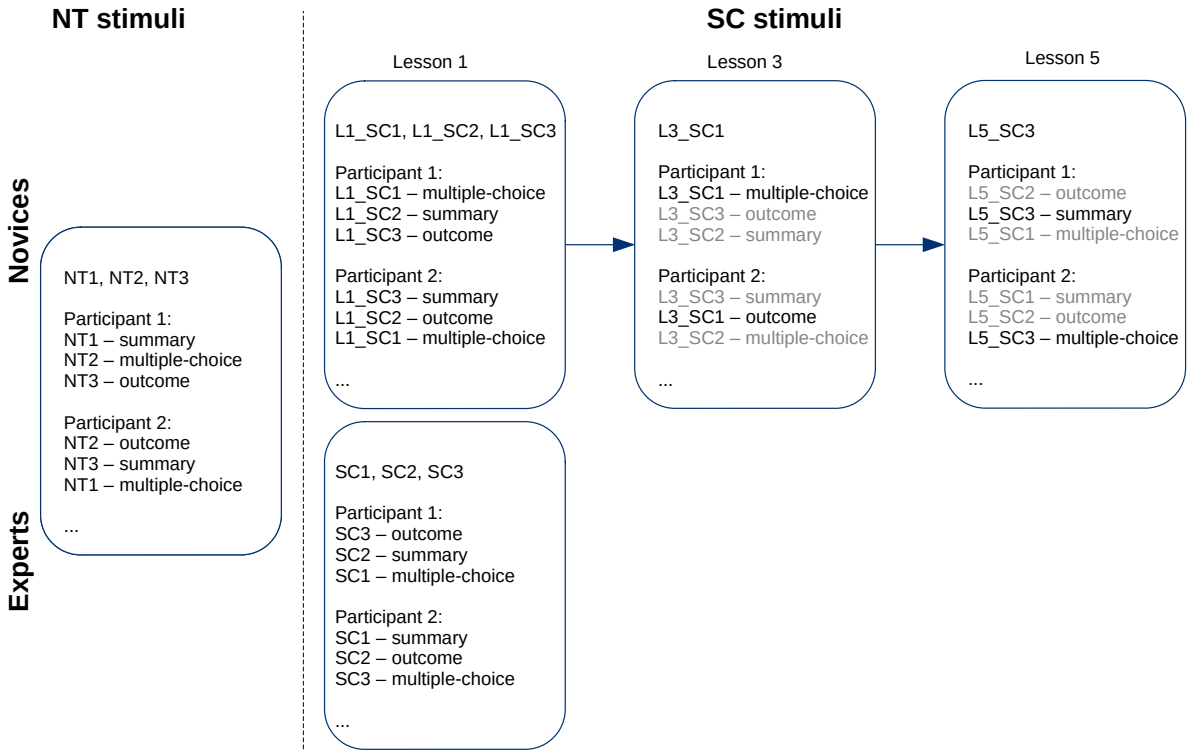


Figure 3.6: Recordings overview: First, both novice and expert participants read three natural-language texts. The novices' gaze on source code was recorded repeatedly, the recordings after lesson 1, 3, and 5 are included in the analysis. Trials that are grayed out are not part of the study. Experts read all texts in a single per participant session.

3.3 Participants

In total, 26 programmers participated in the EMCR study and include novices and professional programmers. Having professionals as experts is a plus factor. Other studies partly use advanced students as expert programmers [Sharafi et al., 2015b, 101], [Sharafi et al., 2020, 3145,3146]. However, comparing early novices to real experts should yield more compelling results. All participants filled in a questionnaire and gave written consent to participate in the study. They were also informed that they can withdraw from the study at any time without consequences.

3.3.1 Novice programmers

The majority of the participants were students, but also two university employees, as well as the relative of a student were drawn to the Java course. Initially, 18 novices started the course, but partly did not attend beyond the first session. Only the ten participants for whom NT and SC data is available are considered for analysis (tables 3.2, 3.4 and figure 3.7). Their age ranged from 20 to 29 years, with a median of 22 years [21..27].[7] Six participants identified as female, four as male. All but one were native German speakers, all indicated to have at least medium English proficiency. Three participants self-reported to have no programming expertise at all, seven had low programming expertise. Eight participants started programming less than one year before taking part in the study. One participant began to learn programming two years prior to the course, but hardly ever actually programmed during that time and never used Java

[7]Median values are given together with 25th and 75th percentile.

before. Another participant attended a programming course seven years before the study, but also hardly ever programmed and had no previous knowledge of Java. With regard to Java, seven participants had no prior knowledge, the remaining three only low expertise. All had less than one year of Java experience. Before attending the Java course, nine participants either programmed not at all or less than one hour per month. Only one participant indicated to program more than one hour per month, but less than one hour per week, both in Java and another language. Seven participants reported to know other programming languages. Except for one medium entry for MATLAB, the expertise in the given languages was rated as low. Two participants disclosed to have myopia, but only one had glasses. All female participants wore eye makeup.

3.3.2 Expert programmers

All expert participants were professional programmers at the time of the recording and have a manifold work background. Several participants worked in an academic field, but the majority in different software companies. They were recruited via personal contacts. In total, 16 expert programmers took part in the study (tables 3.3, 3.5 and figures 3.8, 3.9). Participants' ages ranged from 25 to 49 years, with a median of 30 years [28..37]. Three experts identified as female, 13 as male. Ten participants were native German speakers, two Finnish, and one Chinese, Nepalese, Polish, and Russian respectively. All reported medium to high English proficiency. Nine participants rated their overall programming expertise as high, seven as medium. On average, they started programming 12 years [6..20] prior to the study. Seven experts indicated to have high Java expertise, eight medium, and one low. The participant with low Java knowledge states to have a high overall programming expertise, programs for more than 11 years and more than one hour per day. Additionally, he programmed with Java for about 2.5 years, so he was accepted as expert for the study. On average, participants started using Java 6 years [4..10] before the study. All expert participants program more than one hour per day, either with Java or another language. Most participants spend more time on writing new code than on working on existing one. Furthermore, the majority worked on large projects with more than 50 classes before. All experts have at least basic knowledge of another programming language. How long the participants already have been professional programmers ranges from a few months to 18 years, with a median of 4 years [2..11]. All had normal or corrected to normal vision. Two of the female participants wore eye makeup during the recording.

ID	Age	Gender	Mother tongue	English proficiency	Vision problems	Visual aids	Eye makeup
BR05	28	male	German	high	no	no	no
DO21	22	female	French	medium	light myopia	no	yes
EU10	22	female	German	high	no	no	yes
GO29	22	male	German	high	no	no	no
IO13	20	female	German	medium	no	no	yes
ME23	29	female	German	medium	myopia	glasses	yes
RE11	21	male	German	medium	no	no	no
SA27	24	male	German	high	no	no	no
SE02	28	female	German	high	no	no	yes
SE28	21	female	German	medium	no	no	yes

Table 3.2: Novice participants - general information

ID	Age	Gender	Mother tongue	English proficiency	Vision problems	Visual aids	Eye makeup
AE22	27	male	Finnish	high	no	no	no
BE18	28	male	German	high	no	no	no
BE26	30	female	German	medium	no	no	no
BE29	27	male	German	high	no	no	no
CO20	34	male	German	high	no	no	no
HI27	39	male	German	high	no	no	no
IE30	41	male	German	high	myopia	glasses	no
KK24	25	female	Nepalese	medium	no	no	yes
LK23	29	female	Russian	high	myopia	contact lenses	yes
MR05	26	male	German	medium	no [sic]	glasses	no
PA24	28	male	Finnish	high	no	no	no
RE27	43	male	German	high	myopia	glasses	no
RR04	49	male	German	medium	no	no	no
SI28	32	male	Chinese	medium	myopia	glasses	no
TU15	36	male	Polish	high	no [sic]	contact lenses	no
UL29	31	male	German	medium	no [sic]	glasses	no

Table 3.3: Expert participants - general information

ID	Expertise programming	Years programming	Programming other than Java	Expertise Java	Years Java	Programming Java	Other languages
BR05	none	0	<1 hour/month	none	0	<1 hour/month	–
DO21	low	2	<1 hour/month	none	0	–	low: C++, HTML/CSS
EU10	low	0	–	none	0	–	low: Python
GO29	low	0	<1 hour/month	low	0	<1 hour/month	low: PHP, JavaScript
IO13	low	0	<1 hour/month	low	0	<1 hour/month	low: C++
ME23	low	0	<1 hour/month	low	0	<1 hour/month	low: C++, HTML, PHP
RE11	low	0	<1 hour/week	none	0	<1 hour/week	low: Python, HTML
SA27	none	0	<1 hour/month	none	0	<1 hour/month	–
SE02	low	7	<1 hour/month	none	0	<1 hour/month	low: C; medium: MATLAB
SE28	none	0	–	none	0	–	–

Table 3.4: Novice participants - information on programming experience

ID	Expertise progr.	Years progr.	Progr. other than Java	% Time progr. other than Java	Expertise Java	Years Java	Progr. Java	% Time progr. Java	No. classes biggest Java project	Other languages	Years professional programmer
AE22	high	13	>1 hour/day	70	high	5	<1 hour/month	40	10-50	medium: Object Pascal; high: C, Go, JavaScript	4
BE18	high	11	>1 hour/day	70	low	2.5	<1 hour/month	–	10-50	low: F#; medium: JavaScript, PHP, SQL; high: C#	9
BE26	medium	10	<1 hour/month	50	medium	7	>1 hour/day	50	>50	low: C++, OPAL; medium: C#, SQL	2
BE29	medium	6	<1 hour/week	50	medium	5	>1 hour/day	50	>50	low: Haskell; medium: Bash, C, C++	1
CO20	high	14	<1 hour/day	50	high	12	>1 hour/day	50	>50	low: Python; medium: C, C++	6
HI27	high	20	>1 hour/day	10	medium	20	<1 hour/week	90	10-50	low: Python; medium: JavaScript; high: C, C++	13
IE30	high	30	>1 hour/day	90	high	20	>1 hour/day	90	>50	medium: C++, JavaScript; high: C	15
KK24	medium	5	<1 hour/week	60	medium	1.5	>1 hour/day	40	10-50	low: .NET, PHP	0.5
LK23	high	10	>1 hour/day	90	medium	5	>1 hour/day	90	10-50	low: Scala; medium: C#; high: C++/C4I	5
MR05	medium	5	>1 hour/day	70	medium	2	<1 hour/day	70	10-50	medium: JavaScript, PHP	1
PA24	high	20	<1 hour/week	90	high	10	>1 hour/day	90	10-50	low: C++, R, x86 assembly language; medium: C#, Perl, PIC Assembly language, Visual Basic; high: C, PHP, Python, QBasic	∼ 10
RE27	high	28	<1 hour/month	80	high	13	>1 hour/day	80	>50	low: ABAP; medium: C, C++, JavaScript	15
RR04	medium	25	<1 hour/week	30	high	8	>1 hour/day	50	>50	medium: C; high: C++	18
SI28	medium	6	<1 hour/day	50	medium	6	>1 hour/day	50	>50	low: C, C#; medium: JavaScript	2
TU15	high	20	<1 hour/week	80	high	2	>1 hour/day	80	>50	medium: C, C++, PHP; high: C#	2
UL29	medium	6	<1 hour/week	30	medium	5	>1 hour/day	25	>50	low: C, C++	0.2

Table 3.5: Expert participants - information on programming experience

Figure 3.7: Self-rated programming expertise - novices

Figure 3.8: Self-rated programming expertise - experts

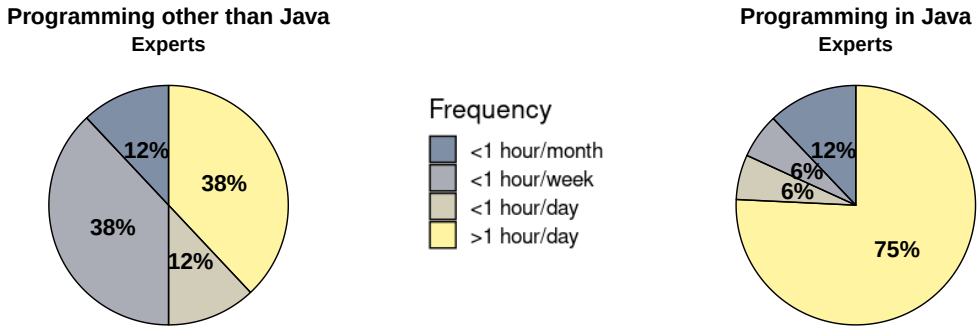


Figure 3.9: Time spent programming - experts

4

Detecting oculomotor events

4.1 Introduction

Even though some analyses can be carried out using raw eye tracking data, e.g. heat maps and scanpath length, the majority of intended analyses necessitate the detection of events in the raw data stream. Events of most interest here and for many other studies are fixations and saccades. Roughly speaking, fixations are events in which the eye is relatively still, while saccades are the movements which occur between fixations [Blignaut & Beelders, 2009, 1], [Holmqvist et al., 2011, 21-23,147,148,150,377], [Karsh & Breitenbach, 1983, 53], [Salvucci & Goldberg, 2000, 71], [Rayner, 1998, 373].

Event detection allows to reduce data complexity, while preserving the information needed for numerous analyses, e.g. by collating microsaccades and other small eye movements into a single event, when they are irrelevant for the study at hand [Salvucci & Goldberg, 2000, 71]. The event detection method has to be chosen carefully, since an inadequate approach may result in erroneous events and can have considerable effects on later analyses [Blignaut, 2009], [Blignaut & Beelders, 2009], [Holmqvist et al., 2011, 149,150,158-160], [Karsh & Breitenbach, 1983, 57-63], [Shic et al., 2008].

4.2 Choosing a suitable approach

During a fixation, the eyes do not stay completely still, but perform several types of small movements, e.g. tremors and microsaccades. The extent of these movements is highly variable [Blignaut & Beelders, 2009, 1-4,10-12], [Holmqvist et al., 2011, 21-23], [Nüssli, 2011, 31-38], [Rayner, 1998, 373,374]. Noise and the imprecision of the recording device also add movement to the data, even when the gaze is predominantly stable [Holmqvist et al., 2011, 161-164]. Hence, classifying raw data samples into periods in which the gaze is predominantly steady (fixations) and periods in which it is moving (saccades) is a complex task. Olsen & Matos [2012, 317] and Inhoff & Radach [1998, 31-34] give examples of ambiguous raw recordings, which make it very difficult to decide on the type of event. The most suitable approach for event detection depends on a number of factors, mainly the eye tracker, its sampling rate and precision, the cognitive task that is studied, as well as the participant [Blignaut & Beelders, 2009, 1-5,10,11], [Holmqvist et al., 2011, 149,154-158,161-164,167,168,181,182], [Karsh & Breitenbach, 1983, 53].

The choice of event detection method can have quite an impact on the accuracy of identified events and in consequence on the analysis results obtained with them. Karsh & Breitenbach

[1983, 57-63] provide an early example on how changing the parameters for fixation detection results in different number of fixations, fixation durations, and scanpaths. Using several algorithms as well as parameter settings, Shic et al. [2008] show that mean fixation duration is mainly a linear function of parameters and that the settings influence the significance of results. Blignaut [2009] confirms these findings for several dispersion metrics and further illustrates that number, position, size, and duration of fixations are in many aspects functions of the chosen thresholds. Similarly, Komogortsev et al. [2010] and Holmqvist et al. [2011, 149,150,158-160] demonstrate the influence of parameters on event detection and how the dependent measures are affected. The various resulting event distributions alter the average and variance of the data and thus the results of all variance-based significance tests, potentially leading to different interpretations of the same set of raw data. Salvucci as cited in Karn [2000, 88] abates that the chosen approach for event detection often only has an effect on part of the data, but does not change the result in general. Nevertheless incorrect events are a threat to validity, so the method and parameters for event detection have to be chosen carefully.
Events can be detected manually [Blignaut, 2009, 882], [Holmqvist et al., 2011, 150], however only algorithmic solutions are considered due to their objectiveness and the huge amount of EMCR data.

4.2.1 Choosing an algorithm

Salvucci & Goldberg [2000] provide a basic taxonomy of fixation identification algorithms, which can serve as basis for systematically comparing and evaluating such algorithms. It classifies algorithms according to their use of spatial and temporal information found in gaze data. There are three spatial criteria: *Velocity-based* algorithms use the velocity of sample points, since fixations have low, saccades high velocities. *Dispersion-based* algorithms assume that during a fixation sample points cluster together and employ the spread distance of samples. *Area-based* algorithms only identify fixations within specified targets. Additionally, there are two temporal criteria: *Duration sensitive* algorithms utilize the duration information of raw samples, *locally adaptive* algorithms allow to take neighboring samples into account, when classifying a point.

Fixations can be identified by dispersion-based algorithms, in which raw data samples must stay within a defined area for at least a certain amount of time, or by velocity-based algorithms, that find sample points with a velocity below a given threshold. During a saccade, the eyes move fast, hence they are usually detected by velocity-based algorithms, which classify sample points above a certain velocity or acceleration threshold as belonging to a saccade. Dispersion-based algorithms are mostly used for data that was recorded at low sampling rates. Velocity-based algorithms on the other hand usually require higher sampling rates in order to get proper velocity and acceleration values. Overall, both dispersion-based and velocity-based algorithms work well and their performance is comparable. Slower eye trackers (ca. <200Hz) cannot reliably detect small saccades and are better suited for fixation identification. These devices and the associated event detection procedures are sometimes called “fixation pickers”. Eye trackers with higher sampling rates and velocity-based detection algorithms are thus tagged “saccade pickers”, since they can robustly detect most saccades [Holmqvist et al., 2011, 151-154,167,168,171,174], [Karn, 2000, 87], [Salvucci & Goldberg, 2000, 77,78].

The EMCR data was predominantly recorded with 120Hz, but a few participants with 60Hz, so event detection will start by identifying fixations using a dispersion-based algorithm. Fixations are also the main type of event later used in the analyses. Dispersion-based algorithms are well-established for event detection. There are several variants, of which the widely-used identification by dispersion threshold algorithm (I-DT) [Salvucci & Goldberg, 2000] will be adopted, which is based on an approach presented by Widdel [1984, 24,25]. The I-DT provides robust results even for somewhat noisy data, despite being a fairly simple algorithm. The two required

parameters are the maximum dispersion of samples belonging to a fixation and minimum fixation duration. Dispersion can be specified in different ways, e.g. as distance between a set of points and their center or between the sample points that are the farthest apart (see 4.2.2.2 *Dispersion*).

During a fixation, samples have a low velocity and cluster closely together. In order to find fixations, the I-DT uses a moving window that spans over the required minimum duration and checks whether the dispersion of the included samples is below the threshold. If the dispersion of these samples exceeds the threshold, the window does not include a fixation (figure 4.1a), and it starts again at the next sample to check the subsequent series of points (figure 4.1b). If dispersion is below the threshold, a fixation is identified and the window is expanded sample by sample until the dispersion surpasses the maximum (figure 4.1c and 4.1d). Finally, the center of the x- and y-coordinates in the window provides the fixation location, the first timestamp in the window the fixation onset, and the time span from first to last sample in the window the duration. The window moves along the samples until the end of the input stream is reached. See figure 4.1 for an example.

time	0	8	16	25	33	41	50	58	67	75
x	507	445	459	459	454	445	443	443	443	342
y	430	456	460	456	458	460	461	461	462	464

(a) The window is initialized with the first contiguous sample points with a minimum duration of 50 ms. The dispersion of these points amounts to 1.7° (71 px), which is above the threshold, so these samples are not part of a fixation.

time	0	8	16	25	33	41	50	58	67	75
x	507	445	459	459	454	445	443	443	443	342
y	430	456	460	456	458	460	461	461	462	464

(b) The window moves to the next series of samples with a duration of at least 50 ms. The dispersion of these points amounts to 0.4° (17 px), which is below the threshold, hence a fixation is found.

time	0	8	16	25	33	41	50	58	67	75
x	507	445	459	459	454	445	443	443	443	342
y	430	456	460	456	458	460	461	461	462	464

(c) The window is expanded, to check, whether the subsequent sample is also part of the current fixation. The dispersion still amounts to 0.4° (17 px), which is below the threshold, so the additional point is included in the fixation.

time	0	8	16	25	33	41	50	58	67	75
x	507	445	459	459	454	445	443	443	443	342
y	430	456	460	456	458	460	461	461	462	464

(d) The window is expanded again, however the dispersion amounts to 2.9° (117 px), which is above the threshold. Therefore the last sample does not belong to the fixation that was just identified.

Figure 4.1: I-DT example using the distance between points that are farthest apart as dispersion metric and a maximum dispersion of 1° of visual angle. At a distance of 65 cm and for an exemplary screen with a height of 29.8 cm and a vertical resolution of 1050 px, 1° corresponds to 41 px. 50 ms is employed as minimum duration. The detected fixation starts at 8 ms, has a duration of 59 ms, and is located at (449,459).

The main disadvantage of the I-DT is the high interdependence of the two parameters, which requires a careful threshold selection. Furthermore, the I-DT, like all dispersion-based algorithms, is not suited for analyzing high-speed gaze data. In some cases the onset of a new fixation is detected while the gaze is still moving and a small distance away from its target. Thereby, most of the allowed dispersion is depleted by points that belong to the end of a saccade. When more samples from the actual fixation are added to the window, even very small intra-fixational movements will cause the dispersion to exceed the threshold and the fixation to end. If the duration of the remaining samples of this fixation is long enough, they will form a new fixation event, which will potentially include the first samples of the subsequent saccade. However, for lower sampling rates, this is not as problematic, since the samples are further apart [Holmqvist et al., 2011, 153,154,167,168].

Details about the algorithm can be found in Blignaut & Beelders [2009], Holmqvist et al. [2011, 153,155,158,159], Nyström & Holmqvist [2010, 190], and Salvucci & Goldberg [2000]. A pseudocode is provided by Salvucci & Goldberg [2000, 74].

4.2.2 Adapting the algorithm

There are several ways of implementing the I-DT and the version presented by Salvucci & Goldberg [2000] is adapted to fit the EMCR data as best as possible. The presented variations are compared in section 4.3 using the EMCR data and diverse parameter settings.

4.2.2.1 Duration

I-DT implementations tend to set the minimal duration in number of samples instead of using an actual duration in milliseconds or the like, e.g. https://github.com/gian/eventdetect, https://github.com/schw4b/emov/tree/master/R.[1]. There are several issues with this method. When working with number of samples, the intended duration threshold can often only be approximated and it depends on the sampling rate and the specific device and recording how well the duration requirement can be met. In theory, a 120Hz tracker provides samples circa every 8 ms. In order to set a minimum duration of e.g. 100 ms, the window has to be initialized with either 12 or 13 samples, which results in an actual threshold of 96 ms or 104 ms - provided samples are truly recorded every 8 ms. This little time difference is pretty much negligible. However, in reality the time span between samples is quite variable. In the 120Hz EMCR data, samples were mostly recorded every 8 or 9 ms, but overall the time span between samples varies between 2 and 5123 ms. In fact, only 64% of the samples are 8 ms apart, another 33% of the samples have a gap of 9 ms. Besides that, 125 other time spans can be found between samples. While those represent only a comparatively small portion of the raw data, they still amount to almost 31,000 samples. Hence, using 12 or 13 samples as threshold, instead of 100 ms, results in diverse minimum window sizes and it is possible that samples will be classified as fixations, even though their duration is shorter than the intended minimum. Likewise, fixations might be overlooked. When samples are slightly further apart or the signal is momentarily lost and samples within a fixation are missing, the number of samples that form a fixation can be less than the minimal required number even though their duration exceeds the intended threshold. Thus, the fixation is not detected, even though it is clearly present in the sample stream. Additionally, when samples are missing, several fixations might be merged into one, if the eye returns to the same area, where it was prior to losing the signal, because the time span between samples is not taken into account (see figure 4.2a). Misclassified samples are not only a local problem, but also influence how successive points are interpreted. Setting the minimal duration as number of samples, instead of as time span, affects the number of fixations as well as fixation durations, since

[1]Last accessed 12/05/2020

some identified fixations are shorter than the intended threshold or have an incorrect duration, and some fixations might be missed altogether.

The I-DT implementation for the EMCR data uses a duration threshold in milliseconds rather than in samples and avoids most of these problems. The impact of this change can be seen in 4.3 *Event detection on the EMCR data*. Nevertheless, even with this duration setting, fixations might be merged into a single event, if there are gaps in the samples during which the gaze moves away and back to the previous area. To prevent this, a time constraint is introduced, which specifies the maximum time span allowed to be missing between two contiguous samples within in a fixation (see figure 4.2). There are several options, how to set such a parameter. It has to be short enough to ensure that the gaze cannot move away, fixate something, and return during the missing time frame. Hence, the missing time has to be less or equal to the minimum fixation duration. It would be possible to allow a slightly longer time span to account for the two saccades that bring the eye away and back, but a cautious choice is preferred. Similarly, Hornof & Halverson [2002, 596] allow a gap of <100 ms within a fixation, which is their minimum fixation duration.

sample	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
time	0	8	16	25	33	41	50	58	67	575	583	592	600	608	617	625	633	642
x	507	445	459	459	454	445	443	443	443	440	440	438	438	435	434	432	430	342
y	430	456	460	456	458	460	461	461	462	461	463	464	463	464	464	465	466	464

(a) The classic I-DT detects a single fixation with a duration of 625 ms.

sample	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
time	0	8	16	25	33	41	50	58	67	575	583	592	600	608	617	625	633	642
x	507	445	459	459	454	445	443	443	443	440	440	438	438	435	434	432	430	342
y	430	456	460	456	458	460	461	461	462	461	463	464	463	464	464	465	466	464

(b) The adapted I-DT, which uses a duration threshold in milliseconds and a time constraint on how far samples can be apart within a fixation, identifies two fixations, the first with a duration of 59 ms, the second with a duration of 58 ms.

Figure 4.2: I-DT example with missing samples. The maximum dispersion is set to 1° of visual angle, minimum duration to 50 ms. The stream contains a gap of 508 ms between samples 9 and 10. It is unknown, where the gaze went during the missing time frame. The gap leaves enough time for the eyes to move away, fixate another area and move back. Furthermore, collating all samples within the dispersion threshold into a single event results in an unusually long fixation, hence treating the samples as two separate events is a more prudent choice.

4.2.2.2 Dispersion

There are a number of different options to specify dispersion. Salvucci & Goldberg [2000, 74] suggest summing the maximum horizontal and vertical distances between points in the window. Other variants include the distance between sample points and their center (i.e. the radius), distance between the points that are the farthest apart, and distance between any two successive points [Blignaut, 2009, 886,887], [Blignaut & Beelders, 2009, 8], [Holmqvist et al., 2011, 155,362,363], [Salvucci & Goldberg, 2000, 74], [Shic et al., 2008, 111]. Blignaut & Beelders [2009] tested several dispersion metrics for the I-DT and found radius and maximum distance between sample points to be the most accurate and least dependent on parameter settings. Since the perceptual span during reading is asymmetric [Holmqvist et al., 2011, 380,381], [Rayner,

1998, 380], the maximum distance between sample points is a better fit for the EMCR data than the radius and therefore implemented as dispersion metric. It ensures that each sample point which belongs to a fixation is within the specified distance to every other point in the fixation. As with other dispersion metrics, the actual area covered by samples belonging to a fixation varies. Shic et al. [2008] and Blignaut [2009] demonstrate that dispersion-based algorithms behave in a comparable manner and can to a certain extent be converted to each other. So it is still possible to relate the events that were identified with this dispersion setting to those detected with other metrics. Unfortunately, it makes event detection more computationally expensive. However this is of no concern for the EMCR study.

Another aspect to consider is that the performance of the I-DT algorithm greatly depends on the quality of the data, especially its spatial precision and the amount of noise. A detailed description about noise in gaze data can be found in 5.2 *Data quality*. The classic I-DT terminates a fixation as soon as dispersion exceeds the threshold, so noise can cause the end of a fixation event and potentially the onset of a new one. One option to account for noise and insufficient precision is to increase the dispersion threshold. An alternative is to allow a certain amount of samples to be outside the maximum spread area [Holmqvist et al., 2011, 155,161-164,181,182], [Nyström & Holmqvist, 2010, 199]. The EMCR data is about reading and contains rather small saccades. Rayner [1998, 373,376] report a mean saccade length of 2° of visual angle for silent reading and that such saccades vary from 1 to over 15 letter spaces. Inhoff & Radach [1998, 32] also mention that very brief saccades with just one character space are common in reading. This strongly limits the margin for increasing the dispersion threshold without risking to collate small saccades into fixations. Instead, it will be allowed that 5% of the samples within a fixation can lie outside the maximum spread area (see figure 4.3). The longer the fixation, the greater the number of samples that can deviate. In order to prevent saccade samples from being included into fixations, the first and last sample belonging to a fixation have to be within the defined dispersion, only samples in between can lie outside. Additionally, if two or more neighboring samples are outliers, their combined duration is not allowed to exceed the minimum fixation duration. The deviating points are not included, when determining the center of the fixation.

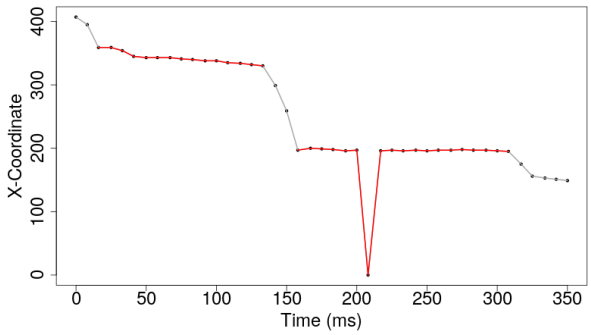


Figure 4.3: I-DT example with one sample outside the maximum allowed spread area. The raw data samples for the x-coordinate are plotted in black, fixations are overlaid in red. At 200 ms, the eye tracker reported a sample with the coordinate (197,452), at 208 ms (0,0), and at 217 ms (196,451). The sample with the location (0,0) exceeds the dispersion threshold and the classic implementation of the I-DT would cut the on-going fixation short. The adapted I-DT leaves the fixation intact.

4.2.3 Setting parameters

In order to compare the data for NT and SC reading, the same settings shall be used for both. The appropriate parameters depend on several factors, like the task that is studied, spatial resolution of the eye tracker, as well as its sampling rate [Blignaut & Beelders, 2009, 5,6], [Holmqvist et al., 2011, 148,154-157], [Karsh & Breitenbach, 1983, 55,59,63], [Salvucci & Goldberg, 2000, 74]. The impact of various parameter options in connection with the different adaptions of the I-DT algorithm can be seen in section 4.3 *Event detection on the EMCR data*.

Since the cognitive task has to be considered, other studies with eye movements in programming were investigated. However, the event detection approach and parameters are rarely stated. In several of the studies that actually describe their approach data was recorded at 300Hz or more and a velocity-based method was used, e.g. Beelders & Plessis [2016a, 3], Fritz et al. [2014, 404], Hansen [2015, 59,60], Lin et al. [2016, 178]. Nüssli [2011, 44-50] developed a velocity-based detection algorithm for 50Hz data, which estimates the optimal threshold. Thus, these cannot serve as reference on parameter settings. Studies with gaze in programming that had a lower sampling rate and use a dispersion-based algorithm mostly state the dispersion threshold in pixels instead of degrees, e.g. Binkley et al. [2012, 224], Fan [2010, 62,68], Uwano et al. [2006, 136]. This makes it difficult to relate the parameters to the EMCR setting, since it depends on the display size and resolution, as well as on the distance between participant and display how many pixels correspond to a degree of visual angle. Since the EMCR data contains NT and SC reading, additionally to parameter settings used in eye movement studies with programming, recommendations for reading were taken into account when choosing the duration and dispersion thresholds.

4.2.3.1 Duration

The suggestions for minimum fixation duration range from about 30 to 250 ms [Blignaut, 2009, 881,884], [Holmqvist et al., 2011, 148-156]. The decision for a certain threshold is often based on the kind of task that is carried out. For instance, fixations tend to be shorter during reading than in picture viewing, so usually a shorter minimum duration is chosen for reading data [Holmqvist et al., 2011, 148,154]. Noted reading research e.g. by Inhoff & Radach [1998, 34] uses 50 ms as minimum duration. Likewise Rayner [1998, 376] and Rayner et al. [2005, 80] emphasize that fixations as short as 50 ms do occur in reading. Furthermore, when the text is obscured after fixating it 50 to 60 ms, reading can continue fairly normal [Rayner et al., 2005, 90]. In studies involving programming, the shorter minimum durations used are 40 ms [Sharif et al., 2012, 382], 50 ms [Uwano et al., 2006, 136] and 60 ms [Sharif et al., 2013, 2]. Yet, some studies discard fixations with a duration of less than 100 ms, e.g. Bednarik & Tukiainen [2006, 127], Fan [2010, 62], Nüssli [2011, 46]. Also many other studies unrelated to programming remove fixations shorter than 100 ms, because they can be the result of noise and it is not certain that visual intake is actually open during such "express" fixations. However, they are still real oculomotor events. Holmqvist et al. [2011, 156] give an example of a fixation even shorter than 50 ms. Hence, these events exist and should be measured properly and not classified as saccades [Holmqvist et al., 2011, 156,157], [Inhoff & Radach, 1998, 34], [Velichkovsky et al., 1997, 514], [Velichkovsky et al., 2000, 80]. Furthermore, the adaption to allow some outliers reduces the chance of spuriously short fixations from noise (figure 4.3). Besides, in code reading such short fixations are most likely used when navigating between different code areas and therefore form a pertinent part of the behavior that is studied. Therefore, in concurrence with well-established reading research, 50 ms is a suitable choice for the EMCR data. In order to better comply with the sampling rates of 60Hz and 120Hz used in the EMCR study, the duration threshold is set to 48 ms, a multiple of 8 and 16 ms.

4.2.3.2 Dispersion

During a fixation the gaze is considered to be stable. However, the eyes do not actually stay completely still, but perform several kinds of small movements. The dispersion threshold defines how much the eye can move, so that the samples are still recognized as fixation. Even though the extent of intra-fixational movements varies among participants, a common threshold can be set, if chosen carefully [Blignaut & Beelders, 2009, 10,11]. Dispersion settings range from 0.5° to 2° of visual angle. For choosing an optimal parameter, spatial resolution, noise, and the task are important factors [Blignaut & Beelders, 2009, 5,6], [Holmqvist et al., 2011, 151-155], [Karsh & Breitenbach, 1983, 55,59,63], [Salvucci & Goldberg, 2000, 74], [Widdel, 1984, 21,22].

In order to lessen the effect of noise, Holmqvist et al. [2011, 163] propose to increase the dispersion threshold. However, the adapted I-DT implementation already takes some noise into account, so this is of minor relevance. Besides, as pointed out in 4.2.1 *Choosing an algorithm*, if the dispersion threshold is set too high, the I-DT might detect the onset of a fixation while the eye is still moving. During silent reading, the average saccade length is 2°, but much shorter saccades are also quite common [Inhoff & Radach, 1998, 32], [Rayner, 1998, 373,376]. Hence, the dispersion setting for the EMCR data has to be well below this amplitude. Blignaut & Beelders [2009] studied thresholds for several I-DT dispersion metrics. For the distance between points farthest apart, which is used for the EMCR data, the range in which the I-DT worked best was found to be 0.9 - 2.2°, with an optimum at 1.3°. Drawing from these prerequisites, the thresholds of 1° and 1.3° were identified as good candidates for the dispersion setting in the EMCR data, since they are within the optimal range, but small enough to not include small saccades into fixations. The EMCR data was recorded with an SMI RED-m eye tracker, which has a spatial resolution of 0.1° (root mean square) and an accuracy of 0.5°. Hence these options do not conflict with the capabilities of the recording device.

Using the adapted I-DT algorithm and a duration threshold of 48 ms, fixations were computed for both potential thresholds. The obtained fixation durations are not normally distributed, so a Mann-Whitney U test was applied for comparison and showed a significant difference between the two sets of fixations ($p<0.001$). The threshold of 1° resulted in more fixations and a lower median fixation duration than 1.3° (see tables 4.2 and 4.3). The x- and y-coordinates of the raw data were plotted with both sets of fixations overlaid. After visual inspection, 1° of visual angle was chosen as dispersion threshold, as it yields better fitting results and reduced the chance of premature fixation onset with the last samples of a saccade. A certain variation in the optimum is to be expected, since Blignaut & Beelders [2009] evaluated the dispersion metric in a study with a completely different stimulus type (chess playing), other participants, and an eye tracker by another manufacturer.

4.3 Event detection on the EMCR data

In order to demonstrate how the chosen approach to event detection behaves in relation to other I-DT variants and parameter settings, various combinations are compared. The dispersion metric used in all variants is distance between the points that are farthest apart. Komogortsev et al. [2010] and Olsen & Matos [2012] use especially constructed fixation and saccade invocation tasks to analyze the effect of different algorithms and settings, which allows them to expect to a certain degree what fixations should be identified and to assess the approaches accordingly. While this is a very good procedure when evaluating such methods in general, here it is not a viable procedure, since the parameters are dependent on task type and the use of I-DT variations is tested specifically for the EMCR data. Consequently, the actual EMCR study data is used for comparing algorithm versions and parameters. In order to have a large data set for testing, the

recordings from novices who eventually quit the study were included, as well as the additional source codes, which participants read, but which are not part of the EMCR stimulus material. The raw data includes 1,653,148 samples from 33 participants (17 novices, 16 experts) and all stimuli texts (212 trials in total).
The I-DT variants included are:

- ms: I-DT with the duration threshold set in milliseconds
- smp: I-DT with the duration threshold set in number of samples
- ms_maxmiss: I-DT with the duration threshold set in milliseconds combined with the time constraint, that during a fixation the interval missing between samples cannot exceed the minimum fixation duration
- ms_out: I-DT with the duration threshold set in milliseconds and 5% of the samples belonging to a fixation are allowed to lie outside the maximum dispersion
- ms_out_maxmiss: I-DT with the duration threshold set in milliseconds combined with the time constraint and 5% of the samples belonging to a fixation allowed outside the maximum dispersion

The tested dispersion thresholds are 0.033, 0.5, 1, 1.3, 1.5, and 2°. Most of these are commonly used and suggested threshold values. The minimum of 0.033° is employed by Komogortsev et al. [2010] when testing different parameter settings on dispersion-based algorithms and represents the smallest amplitude usually assumed for intra-fixational saccades. 1° and 1.3° were identified as the best candidates for the EMCR data. The settings of 32, 48, 96, 144, 192, and 240 ms are employed for minimum fixation duration. All duration thresholds were chosen to be multiples of 16, since samples should be recorded approximately every 8 ms with the 120Hz device and every 16 ms with the 60Hz one. The shortest duration used in other studies was 40 ms and 32 ms is the closest multiple of 16. Besides, fixation durations of 30 to 40 ms are possible [Holmqvist et al., 2011, 381]. 48 ms is the parameter selected for event detection, it is as close to 50 ms as possible. 96, 144, 192, and 240 ms correspond to 100, 150, 200, and 250 ms, the settings often suggested in literature.

The I-DT only detects fixations and does not classify the remaining samples. Usually, the samples between fixations are taken as saccades [Karn, 2000, 87], [Nyström & Holmqvist, 2010, 190]. This approach is also adopted here. Saccade duration is calculated as time span between the end of the preceding fixation and the begin of the subsequent one. Even though saccades often do not take the shortest path between two points, but can undergo several shapes and curvatures [Holmqvist et al., 2011, 23], saccadic amplitude is calculated as Euclidean distance between the locations of two fixations, an often adopted practice, e.g. [Holmqvist et al., 2011, 311,319], [Nyström & Holmqvist, 2010, 200]. The dataset for demonstrating the differences between the I-DT variants and parameter combinations includes samples in which the eye tracker reported a gaze location of (0,0), i.e invalid samples. While events from such samples will not be used in the analysis, they were not removed from the following comparison in order to show the exact behavior of the approach with the actual data. Excluding certain events would not affect all combinations in the same way and thus bias the outcome of the comparison.

4.3.1 Comparing algorithm variants

The number of identified fixations, as well as the minimum, median, and maximum fixation durations for all I-DT variants are summarized in tables 4.2 and 4.3. A Kruskal-Wallis test was used to compare the fixation durations yielded by all variants for the chosen threshold of 1° and 48 ms, which showed a significant difference between them ($p<0.001$). Therefore pairwise comparisons were carried out with Mann-Whitney U tests. The resulting p-values were corrected for multiple testing (table 4.1).

	smp	ms_maxmiss	ms_out	ms_out_maxmiss
ms	0.069	0.984	**<0.001**	**<0.001**
smp	-	0.069	**<0.001**	**<0.001**
ms_maxmiss	-	-	**<0.001**	**<0.001**
ms_out	-	-	-	0.876

Table 4.1: Differences between the fixation durations generated by the I-DT variants for the chosen thresholds of 1° and 48 ms.

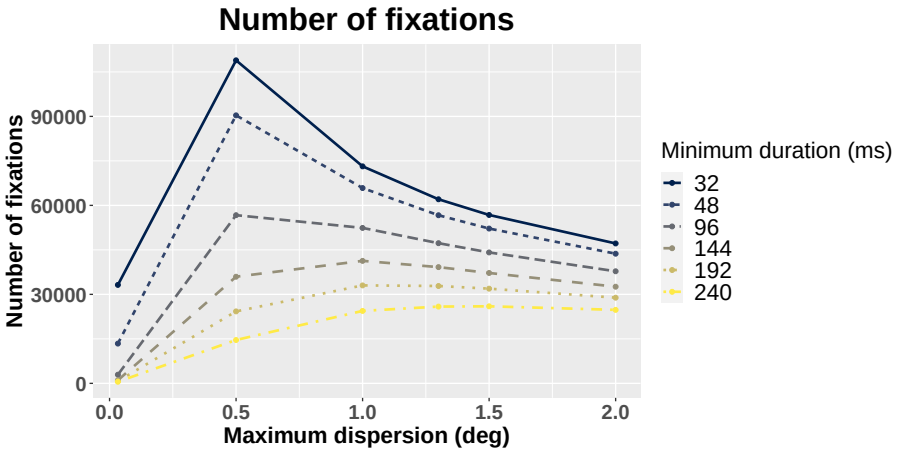
When comparing the classic I-DT with minimum duration in milliseconds (*ms*) and in number of samples (*smp*), it becomes obvious that for all parameter combinations more fixations are found by the *ms*-version, even though the algorithms are identical, except for how the minimum duration is specified. Thus, the *smp*-variant actually misses some fixations. Furthermore, it indeed found fixations that are shorter than the intended threshold. Despite these issues, there is no significant difference between *ms* and *smp*, though the resulting p-value is only 0.069, which suggests that the two ways of specifying minimum duration may have a subtle effect. Nevertheless, the *smp*-variant is not considered further, since the anticipated problems with a threshold set in number of samples did actually occur. Introducing a maximum time span that can be missing between fixation samples (variants *ms_maxmiss* and *ms_out_maxmiss*) seriously decreases the maximum fixation duration for half of the parameter combinations. No differences were found between *ms* and *ms_maxmiss* (p=0.984), *smp* and *ms_maxmiss* (p=0.069), and *ms_out* and *ms_out_maxmiss* (p=0.876). Consequently, the time constraint does not change the outcome in general, but removes some of the implausibly long fixations. This is the optimal result. The two I-DT adaptions which allow some samples outside the dispersion threshold (*ms_out* and *ms_out_maxmiss*) are significantly different from the other variants (<0.001). They generally result in fewer fixations and longer median fixation durations.

4.3.2 Comparing parameter variants

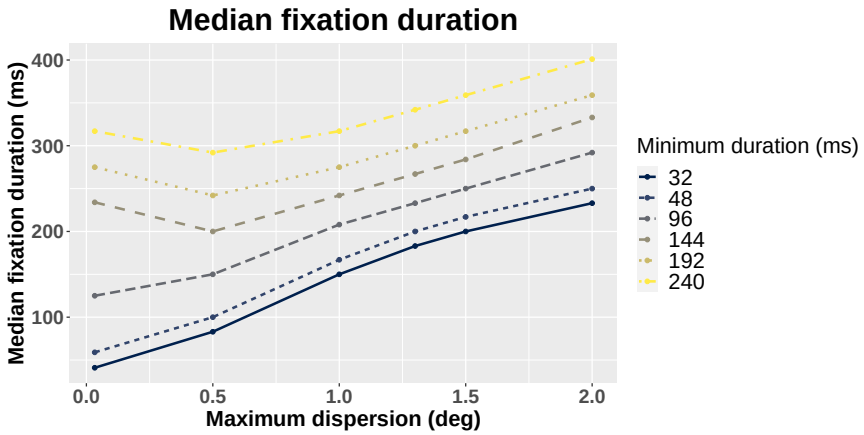
In order to show how the I-DTs behave for different parameter combinations, tables 4.2 and 4.3 list the results for all variants and parameters. However only the approach *ms_out_maxmiss*, which is used for the EMCR study, is examined in detail.

Changing the thresholds affects the number of identified fixations, as well as minimum, median, and maximum fixation duration. Combining a small dispersion threshold with a long minimum duration, often prevents the I-DT from finding any fixations and thus results a very small overall number of fixations. Focusing on dispersion values of $\geq 0.5°$, which can reasonably be used as thresholds, shows that as dispersion increases, the number of identified fixations converges for all duration thresholds, while the median fixation durations retain their uniformly increasing slope (see figure 4.4).

Inspecting the results for the threshold setting of 1° and 48 ms shows that the identified events are within the expected range. According to Holmqvist et al. [2011, 381], fixations usually last around 200 to 300 ms, but may have a range from about 30 ms to several seconds. Furthermore, average fixation durations vary across different tasks and stimuli, e.g. Rayner [1998, 373,376] gives a mean fixation duration of 225 ms for silent reading and points out that there is huge variability between readers both for fixation duration and saccade length. The median fixation duration for the EMCR data is 167 ms [92..267], the maximum is about 4.5 s, which is much more realistic than the maximum durations of over 12 sec found by other variants and parameter combinations. Nevertheless, it is very reassuring that the adapted I-DT together with the chosen thresholds only generated physiologically reasonable fixations, which is not the



(a) Number of detected fixations



(b) Median durations of detected fixations

Figure 4.4: Fixations detected by the I-DT variant *ms_out_maxmiss* for different parameter combinations

case for many of the other combinations. As expected, the distribution of fixation durations has a positive skew (figure 4.5).

With the rationale that fixations last much longer than saccades and therefore a substantial part of the samples should be attributed to fixations, Nüssli [2011, 39,40] uses the total fixational coverage as part of a fixation identification quality score. Similarly, Blignaut [2009] employs the percentage of samples included in fixations as one indicator for finding optimal dispersion thresholds. Using the chosen approach, 91% of the samples are part of a fixation, thus a very huge portion of the raw data is collated into fixations. The remaining samples constitute saccades, noise and other events that are not classified. As additional check that the chosen approach works properly, saccades were calculated and inspected. The median saccade velocity is 116°/s [57..200], which is well in accordance with literature. Nevertheless, some of the samples that are classified as saccades might actually be noise.

4.3.3 Post-Processing

Irrespective of the adopted approach, there are several steps of processing the detected events before analysis. If consecutive fixations are located very near to each other, they might actually be part of one longer fixation that was split due to large intra-fixational movements or noise.

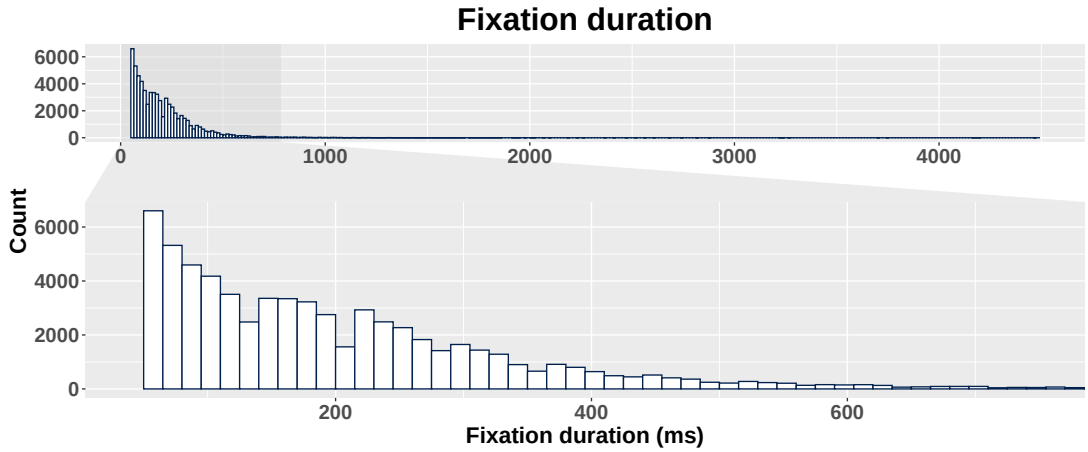


Figure 4.5: Distribution of fixation durations in ms. Fixations were identified by the I-DT variant *ms_out_maxmiss* using the parameters of 1° and 48 ms.

Such close fixations can be merged after event detection [Komogortsev et al., 2009, 2636,2637]. In order to make sure that only those fixations are combined for which the disruption is absolutely evident, a very guarded merging is applied. Fixations are only collated when their spatial distance is less than 0.1° (ca. 5 px) and less than 48 ms pass between them. The onset of the collated fixation is the first sample of the two original fixations, the center point is calculated from all included sample points. This additional processing reduces the number of fixations for the chosen parameters from 65841 to 65639, hence only 0.3% of the fixations in the EMCR data were merged into others. The fixation durations before and after merging were compared with a Mann-Whitney U test, but no difference was found (p=0.369). Thus, the merging does not change the distribution of fixation durations, but joints fixations that were obviously cut short.

When an event spans over the transition from one stimulus to the next, the first raw data samples of this event are recorded during the earlier trial and are therefore missing in the subsequent one. Hence the first event in the later trial appears shorter than it actually is, potentially distorting data analysis. Such incomplete events can be excluded from analysis [Holmqvist et al., 2011, 154,384,385]. Consequently, the first event of each trial of the EMCR data is discarded. The slide presented before the stimulus text is the instruction and participants have to press the left mouse button in order to get to the text stimulus. After that, they shift their attention back to the screen and look about for the text to be read. So in any case, the first event is rather irrelevant for text comprehension. Analogous, the last event of a trial is not included into the analysis, as it might be split as well.

Furthermore, it is possible to filter out some noise by discarding samples with a physiologically impossible velocity [Holmqvist et al., 2011, 152,181]. Similarly to Nyström & Holmqvist [2010, 193], saccades are removed, when their velocity exceeds 1000°/s. Finally, fixations with the location (0,0) are discarded together with the saccades leading to and coming from them. Such samples are not removed before event detection, since that would leave gaps in the raw data and the missing time would mostly be attributed to saccades, distorting their distribution.

4.4 Conclusion

Choosing a suitable event detection approach is crucial for obtaining valid event locations and durations. Many methods exist for event detection and their suitability strongly depends on the data that is to be processed. There are still challenges and event detection is an active research topic, e.g. Friedman et al. [2018], Korda et al. [2018], Zemblys et al. [2018]. Nevertheless, the adapted I-DT is a solid choice for the EMCR data. It is widely used and provides accurate and robust results for fixation events, which are the main focus of the EMCR study. Modifications like using the maximum distance between sample points as dispersion metric, setting the dispersion threshold in milliseconds, restricting the time allowed to be missing between samples within a fixation, and allowing a certain amount of samples outside the dispersion threshold attenuate several of the I-DT's drawbacks. For detecting fixations in the EMCR data the maximum dispersion is set to 1° of visual angle and 48 ms is employed as minimum fixation duration.

max_disp	min_dur	ms	smp	ms_maxmiss	ms_out	ms_out_maxmiss
0.033	32	33354	32614	33349	33176	33192
0.033	48	13521	13095	13539	13374	13409
0.033	96	2468	2264	2504	2886	2927
0.033	144	972	905	975	1049	1052
0.033	192	691	631	690	747	746
0.033	240	492	446	490	538	536
0.5	32	112570	111806	112433	108987	108908
0.5	48	93540	92767	93496	90362	90361
0.5	96	55492	54710	55498	56689	56700
0.5	144	34753	34129	34744	35958	35949
0.5	192	22415	20454	22406	24272	24263
0.5	240	12935	11500	12928	14622	14615
1	32	76718	76228	76640	73146	73162
1	48	69036	68575	68991	65817	65841
1	96	52879	52316	52877	52396	52402
1	144	41442	40958	41425	41315	41298
1	192	32092	30425	32074	33042	33024
1	240	23186	21548	23167	24442	24422
1.3	32	65345	64934	65301	61992	62062
1.3	48	59718	59338	59675	56641	56678
1.3	96	48156	47677	48147	47247	47259
1.3	144	39738	39323	39716	39204	39188
1.3	192	32369	31046	32349	32841	32824
1.3	240	24984	23662	24963	25899	25878
1.5	32	59943	59558	59930	56655	56770
1.5	48	55103	54741	55068	52121	52190
1.5	96	45147	44709	45139	44122	44138
1.5	144	37984	37631	37963	37229	37213
1.5	192	31748	30595	31728	31947	31925
1.5	240	25338	24205	25321	25989	25972
2	32	50080	49723	50106	46987	47168
2	48	46488	46155	46465	43605	43715
2	96	38995	38609	39010	37748	37787
2	144	33602	33311	33590	32600	32590
2	192	29207	28378	29188	28905	28885
2	240	24613	23798	24593	24804	24779

Table 4.2: Number of fixations identified by the different I-DT adaptions:
ms: duration threshold in milliseconds
smp: duration threshold in number of samples
ms_maxmiss: duration threshold in milliseconds with time constraint for the interval between fixation samples
ms_out: duration threshold in milliseconds, 5% of the samples belonging to a fixation are allowed outside the maximum dispersion
ms_out_maxmiss: duration threshold in milliseconds with time constraint and 5% of the samples belonging to a fixation allowed outside maximum dispersion
The maximum dispersion is given in degree of visual angle, minimum fixation duration in milliseconds. The chosen thresholds of 1° of visual angle and 48 ms are highlighted.

		ms			smp			ms_maxmiss			ms_out			ms_out_maxmiss		
max_disp	min_dur	min	median	max	min	median	max	min	median	max	min	median	max	min	median	max
0.033	32	32	41	11922	30	42	11922	32	41	3663	32	41	11963	32	41	4205
0.033	48	49	59	11922	49	59	11922	49	59	4186	49	59	12038	49	59	4463
0.033	96	99	125	11922	99	133	11922	99	125	4591	99	125	12038	99	125	4591
0.033	144	150	242	11922	150	250	11922	150	241	11922	150	234	12038	150	234	12038
0.033	192	192	275	11922	199	284	11922	192	275	11922	192	275	12038	192	275	12038
0.033	240	241	316	11922	250	325	11922	241	312	11922	241	317	12038	241	317	12038
0.5	32	32	83	11922	29	83	11922	32	83	3663	32	83	11963	32	83	4205
0.5	48	48	100	11922	48	100	11922	48	100	4186	48	100	12038	48	100	4463
0.5	96	97	150	11922	95	150	11922	97	150	4591	97	150	12038	97	150	4591
0.5	144	146	200	11922	149	200	11922	146	200	11922	146	200	12038	146	200	12038
0.5	192	192	242	11922	199	250	11922	192	242	11922	192	242	12038	192	242	12038
0.5	240	241	284	11922	248	300	11922	241	284	11922	241	292	12038	241	292	12038
1	32	32	142	11922	32	142	11922	32	142	4164	32	150	11963	32	150	4205
1	48	49	159	11922	44	159	11922	49	159	4186	49	167	12038	49	167	4463
1	96	98	200	11922	98	201	11922	98	200	4591	99	208	12038	99	208	4591
1	144	149	234	11922	149	242	11922	149	234	11922	149	242	12038	149	242	12038
1	192	192	275	11922	197	283	11922	192	275	11922	192	275	12038	192	275	12038
1	240	241	317	11922	249	326	11922	241	317	11922	240	317	12038	240	317	12038
1.3	32	33	167	11922	33	175	11922	33	167	4456	33	183	11963	33	183	4447
1.3	48	49	191	11922	49	192	11922	49	191	4456	49	200	12038	49	200	4463
1.3	96	98	226	11922	98	233	11922	98	226	4591	98	233	12038	98	233	4591
1.3	144	148	259	11922	148	259	11922	148	259	11922	148	267	12038	148	267	12038
1.3	192	192	292	11922	197	300	11922	192	292	11922	192	300	12038	192	300	12038
1.3	240	241	334	11922	249	350	11922	241	334	11922	241	342	12038	241	342	12038
1.5	32	32	184	11922	33	184	11922	32	184	3930	32	200	11963	32	200	5015
1.5	48	49	201	11922	45	201	11922	49	201	4186	49	217	12038	49	217	5015
1.5	96	98	242	11922	99	242	11922	98	242	4591	98	250	12038	98	250	5015
1.5	144	149	275	11922	149	276	11922	149	275	11922	149	284	12038	149	284	12038
1.5	192	192	309	11922	197	317	11922	192	309	11922	192	317	12038	192	317	12038
1.5	240	241	351	11922	249	367	11922	241	351	11922	241	359	12038	241	359	12038
2	32	32	217	11922	31	217	11922	32	217	6884	32	233	11963	32	233	6884
2	48	49	234	11922	49	234	11922	49	234	6884	49	250	12038	49	250	6884
2	96	99	283	11922	99	283	11922	99	283	6884	99	292	12038	99	292	6884
2	144	149	317	11922	149	317	11922	149	317	11922	149	333	12038	149	333	12038
2	192	192	351	11922	195	359	11922	192	351	11922	192	366	12038	192	359	12038
2	240	241	400	11922	249	409	11922	241	400	11922	241	401	12038	241	401	12038

Table 4.3: Fixation durations generated by the different I-DT adaptions: The maximum dispersion is given in degree of visual angle, all other values are in milliseconds. The chosen thresholds of 1° of visual angle and 48 ms are highlighted.

5 Eye tracking error

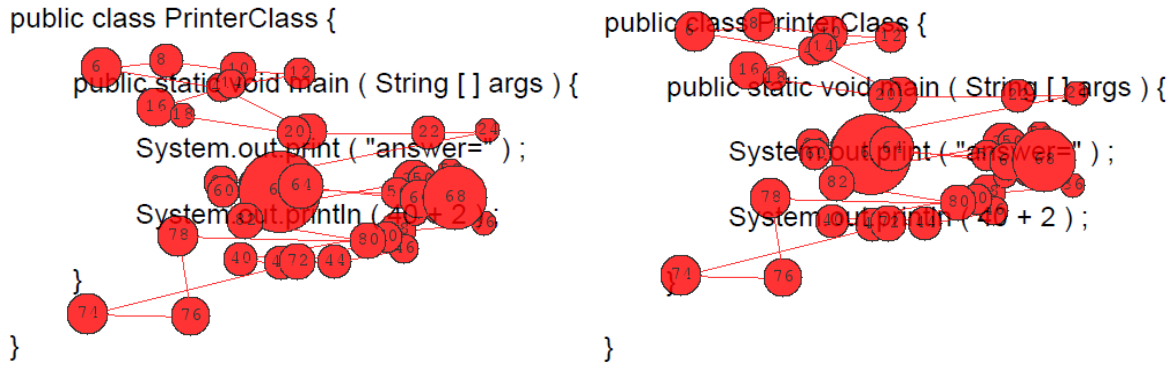
5.1 Introduction

Eye movement data often exhibits a disparity between the true gaze position and the recorded one. This issue is widely acknowledged, e.g. by Feit et al. [2017], Holmqvist et al. [2011, 224,225], Holmqvist et al. [2012, 45-47], Hyrskykari [2006], Martínez-Gómez & Aizawa [2014, 95], Nevalainen & Sajaniemi [2004, 156], Niehorster et al. [2018], Nüssli [2011, 50-67], Nyström et al. [2012, 272,273], Špakov et al. [2018].

Whether spatial error is problematic depends on the purpose of the data. For stimuli with big AOIs or sufficient margin space, like a large button in gaze interaction, a certain offset does not pose a problem. In reading research however, AOIs are usually small and close together [Carl, 2013, 1], [Holmqvist et al., 2012, 45], [Nyström et al., 2012, 272], [Zhang & Hornof, 2011, 834]. Hence, errors can result in assigning the gaze to the wrong part of the stimulus, thereby distorting a great many measures used for analysis, heatmaps, and scanpaths (see figure 5.1 for an example). Accordingly, many reading studies only employ isolated words or sentences [Martínez-Gómez & Aizawa, 2014, 95]. Using dwell time, as well as number and duration of fixations as example, Holmqvist et al. [2012, 45-47] demonstrate that already a rather small offset can lead to false conclusions. Therefore, in order to obtain reliable results, the error needs to be addressed before analysis [Holmqvist et al., 2012, 45], [Lohmeier, 2015, 33], [Nüssli, 2011, 50,51,64,67], [Špakov et al., 2018, 2], [Zhang & Hornof, 2011, 834]. This is not an easy task, since error can vary over space and time [Feit et al., 2017, 1120-1122], [Holmqvist et al., 2011, 42], [Holmqvist et al., 2012, 47], [Hornof & Halverson, 2002, 600-602], [Lohmeier, 2015, 35], [Nyström et al., 2012, 276, 281,282,285], [Zhang & Hornof, 2011, 840,841], [Zhang & Hornof, 2014, 95,96]. However, even when it is obvious where the data belongs, modifications always include the risk of producing false results, may it be deliberate or not [Holmqvist et al., 2011, 225]. Also, despite being a common problem in eye tracking data, studies often either do not address error in their data or at least do not report error handling. Partly, the impact of errors might be underestimated. In any case, when it is unclear whether error occurred and was treated sufficiently, data and results are of limited value [Hornof & Halverson, 2002, 593], [Wyatt, 2010, 1985], [Zhang & Hornof, 2011, 841,842].

> "It is much easier to simply report the eye tracker accuracy given by the manufacturer and, from then on, to ignore any possible error in the eye-tracking data or, if error happens to be noticed on some trials, to just discard those trials. However, we believe that a bold, daring, and honest look at eye movement data and a commitment to attacking error is critical for the advancement of eye-tracking research and application." [Zhang & Hornof, 2011, 842]

Eye tracking error and how it can be addressed is discussed in more detail in the next sections.



(a) Uncorrected (b) Corrected

Figure 5.1: Scanpaths

5.2 Data quality

5.2.1 Accuracy, precision, valid samples

Any domain needs good data quality to allow for valid research results. For eye movement data, key aspects of quality are the eye tracker's accuracy and precision, as well as the amount of valid samples [Holmqvist et al., 2011, 29,33], [Holmqvist et al., 2012, 45-48], [Nyström et al., 2012, 272,273,283]).

> "When the eye tracker reports a valid sample, data quality can be defined as the distance θ_i (in visual degrees) between the actual x_i and the measured $\hat{x}_i$ gaze position, known as the *spatial accuracy*, or just accuracy, as well as the difference between the time of the actual movement of the eye t_i and the time reported by the eye tracker $\hat{t}_i$, known as *latency* or *temporal accuracy*. If both accuracy and precision differences are zero, the data quality for this single sample is optimal." [Holmqvist et al., 2012, 48]

The *accuracy* of an eye tracker is defined as the (average) distance between the true gaze position and the measured one. Accuracy is usually best right after calibration, but decreases during the recording. Good accuracy is crucial for analyses using AOIs [Holmqvist et al., 2011, 33,41,42], [Holmqvist et al., 2012, 45], [Hornof & Halverson, 2002, 593], [John et al., 2012, 297], [Nyström et al., 2012, 272,276,285,286].

Precision is the degree in which measurements under the same conditions reliably reproduce a result [Holmqvist et al., 2011, 33,34], [Holmqvist et al., 2012, 46], [John et al., 2012, 297], [Nyström et al., 2012, 273]. Spatial precision denotes the variance in accuracy, temporal precision the variance in latency. Latencies chiefly occur between the actual eye movement and the

moment it is recorded, as well as between stimulus presentation and recording software. Temporal precision is especially critical for studies including gaze-contingency or synchronization to external equipment [Holmqvist et al., 2011, 33,43-47], [Holmqvist et al., 2012, 48].

For the study presented in this work only spatial error poses a major threat to validity and will be addressed in detail. Accuracy and precision are influenced by several types of noise: system-inherent, oculomotor, and environmental noise, as well as optic artifacts [Holmqvist et al., 2011, 33,34,117,118]. In this context, two types of error are differentiated (see figure 5.2). A *systematic error* or bias is a quite constant deviation between measured and true gaze location, indicating low accuracy. If the recorded gaze is spread around the true gaze point, it is called *variable error*, suggesting low precision. A systematic bias is easier to detect and correct than randomly distributed deviations with a mean around zero. Both types of error can occur simultaneously [Chapanis, 1951, 1181,1182,1187-1190], [Hornof & Halverson, 2002, 592,593], [Lohmeier, 2015, 34,35].

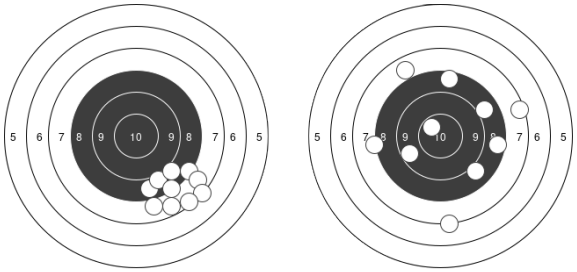


Figure 5.2: Error types: systematic (left) and variable (right), based on Chapanis [1951, 1181]

Invalid samples occur when the eye cannot be reliably recorded. This usually happens when essential components of the eye are obscured, e.g. by glasses or blinks, or if something else is mistakenly detected as eye feature [Holmqvist et al., 2011, 117,118], [Holmqvist et al., 2012, 47,48], [Nyström et al., 2012, 272,273].

5.2.2 Error sources

Errors in eye tracking data originate from a variety of sources in the recording system's hard- and software, the recording environment, as well as the participants. They also occur together. Some issues, like mascara are already problematic during calibration, others only arise during recording, e.g. head movements [Carl, 2013, 3], [Holmqvist et al., 2011, 37,117,118], [Holmqvist et al., 2012, 47], [Hyrskykari, 2006, 661], [Nyström et al., 2012, 273,281-285], [Zhang & Hornof, 2014, 95]. Below, a number of relevant factors are outlined, which affect at least one, but often several aspects of data quality.

5.2.2.1 Factors concerning the recording system and environment

Depending on hardware properties like camera resolution, the recording device has a limit to how exact it can measure the gaze. Furthermore, choosing appropriate algorithms and parameters, e.g. velocity- or dispersion-based fixation detection, strongly influences the computed gaze locations. Also, experimental conditions are rarely optimal, hence the device's actual performance often falls below its potential [Carl, 2013, 3], [Holmqvist et al., 2011, 37,41], [John et al., 2012, 297], [Niehorster et al., 2018], [Nyström et al., 2012, 272].

Factors affecting data quality include, but are not limited to:

Type of system: Tower-mounted eye trackers usually provide better accuracy than head-mounted systems and remote devices [Holmqvist et al., 2011, 42].

Camera resolution: In order to achieve good precision, the camera has to capture the pupil with as many pixels as possible, thus needs a high resolution [Holmqvist et al., 2011, 37,38], [Holmqvist et al., 2012, 47].

Sampling frequency: The sampling frequency determines to what extent fast movements can be captured. It needs to be high in order to record short events. With low sampling frequencies outliers can have quite a negative impact [Holmqvist et al., 2011, 29-32], [Holmqvist et al., 2012, 47,50].

Calibration method: Participant-controlled calibration seems preferable over operator- and system-controlled calibration, partly because the participant is better able to decide when the eye is stable and focused on the target than the operator or the software [Nyström et al., 2012, 274-276,281-284].

Recording monocularly or binocularly: Using both eyes can increase accuracy and precision, especially for dispersion-based fixation detection. On the other hand, averaging the samples from both eyes is problematic for determining saccade measures or when one eye is lost [Holmqvist et al., 2011, 59,60], [Holmqvist et al., 2012, 47].

Light conditions: If other light sources than the illumination from the recording system are present, like the sun or indoor lights, additional reflections might appear on the eye or the contrast in the eye image is reduced. Also, changing luminance either in the environment or from the stimulus itself can result in recording false movements [Drewes et al., 2012, 209,210], [Holmqvist et al., 2011, 40,125,126], [Holmqvist et al., 2012, 47].

Movements: Movements near the eye tracker can reduce the stability of the signal, e.g. clicking a mouse on the same table as the eye tracker [Holmqvist et al., 2011, 35,40], [Holmqvist et al., 2012, 47]. Furthermore, for head-mounted eye trackers the device can slip on the head, thereby changing the relation between eye and camera [John et al., 2012, 297].

Viewing distance: In order to achieve high precision, the camera needs to capture a good image of the pupil. Hence, precision decreases with greater distance between camera and eye [Holmqvist et al., 2011, 37,38,59,], [John et al., 2012, 297].

Stimulus position: Data quality is usually best for targets in the center of the screen. Reasons for the decreasing quality towards the edges can be eyelashes, which obscure the eye image, and the reduced pupil size due to the larger viewing angle [Feit et al., 2017, 1122], [Holmqvist et al., 2011, 42], [Holmqvist et al., 2012, 47], [Hornof & Halverson, 2002, 600-602], [Nyström et al., 2012, 281,282,285], [Zhang & Hornof, 2011, 841], [Zhang & Hornof, 2014, 95,96].

Expertise of the operator: More experience with eye trackers and especially the type and model in use is beneficial for data quality, since the operator knows how to optimize the setup and to instruct participants [Holmqvist et al., 2012, 47,51], [Nyström et al., 2012, 276,284,285].

5.2.2.2 Factors concerning the participant

The quality of the eye image and how well it is possible to detect features from it strongly determines the quality of eye tracking data. On the part of the participant, there are a great many characteristics that interfere with the eye image [Nyström et al., 2012, 285]. Furthermore, even with a highly exact recording, the size of the fovea sets a limit to recording accuracy. It covers about 1.5 - 2° of the visual field, which at a standard recording distance of 70 cm corresponds to an area with a diameter of roughly two cm. Hence, it is enough to fixate a location close to the object of interest in order to see it, which will result in an offset between

the recorded gaze location and the AOI [Holmqvist et al., 2011, 42], [Holmqvist et al., 2012, 48].

Factors affecting data quality include, but are not limited to:

Eye physiology: A number of characteristics in the individual's eye physiology are unfavorable for recording high quality data. Long or downward-pointing eyelashes, as well as droopy eyelids can cover the pupil. Eye color is also relevant. Data from bluish eyes tends to be less precise than from brown or other colored eyes. Furthermore, data quality depends on how well the actual eye corresponds to the model in the recording software [Holmqvist et al., 2011, 41-43,57,118,120-122], [Holmqvist et al., 2012, 47], [Nyström et al., 2012, 276,281,282,285].

Visual aids: Glasses pose multiple challenges for recording, e.g. by causing additional reflections in the eye image or due to the rim obscuring parts of the eye. Contact lenses are also problematic. They can shift during the recording or have air bubbles underneath, which leads to several smaller reflections [Holmqvist et al., 2011, 29,34,43,57,118,122-125], [Holmqvist et al., 2012, 47], [Nyström et al., 2012, 274,276,281,282,285].

Eye makeup: Mascara and other makeup can make it difficult to detect eye features by obscuring the image. Furthermore, the dark area can be mistaken for the pupil [Holmqvist et al., 2011, 29,119-121], [Holmqvist et al., 2012, 47].

Pupil size: The center of a big pupil is easier to detect, hence both accuracy and precision are better for larger pupils. When pupil diameter changes during the recording, the center of the pupil shifts relative to the eye and the eye tracker may mistakenly report an eye movement, even though the gaze remains steady. Pupil dilation or constriction are mostly induced by changes in stimulus and ambient brightness, but even under constant light conditions, cognitive or emotional processes, as well as the viewing angle can have an impact [Drewes et al., 2012, 209,210], [Gagl et al., 2011, 1171,1172,1174], [Holmqvist et al., 2011, 43,391-394], [Holmqvist et al., 2012, 47], [Nyström et al., 2012, 281,285], [Wyatt, 2010, 1982,1985-1987].

Movements: Some setups allow the participant to move their head and body. Even a small movement of two cm to or from the screen can increase the error considerably [Carl, 2013, 1,3], [Cerrolaza et al., 2012, 205,208], [Holmqvist et al., 2011, 39,43,51,52], [Holmqvist et al., 2012, 47], [Niehorster et al., 2018], [Nyström et al., 2012, 273,285,286].

Behavior: Participants exhibit a wide range of behaviors, that are problematic for data quality. For example, some open their eyes and tense during calibration. During the recording they relax, change their position, or close their eyes slightly [Holmqvist et al., 2011, 42], [Holmqvist et al., 2012, 47].

5.2.3 Conclusion

Eye tracking data should be as accurate and precise as possible and consist of valid samples. However, often there is a disparity between the reported gaze position and the actual one, which can arise from a number of different sources. Several factors already prevent a good calibration, the prerequisite for high quality data. Others cause a deterioration during the recording. Some error sources can be lessened to a certain degree e.g. by optimizing the recording system and environment. Others, like the participant's physiological characteristics are difficult to tackle.

5.3 Existing approaches for addressing error

Error in eye tracking data is difficult to avoid. How it can be dealt with depends on the degree of exactness required for the intended analysis. Preventing problematic data in the first place is only possible in some cases, so the data usually has to be either discarded or corrected. Figure 5.3 outlines the relevant approaches to address error in gaze data, which are discussed below with regard to the EMCR study.

Offsets between true and recorded gaze location are not just problematic for the validity of research results, but also for gaze-based interaction, e.g. by making it difficult to select a button with gaze [Holmqvist et al., 2012, 46]. There are several techniques to tackle inaccuracy in gaze interaction, e.g. Ashmore et al. [2005], Bates & Istance [2002], MacKenzie & Zhang [2008], Miniotas et al. [2004], Skovsgaard et al. [2010], Vidal et al. [2013], and Zhang et al. [2008]. However, these do not apply here and are therefore not considered further.

5.3.1 Error debilitation and data removal

Strategies to deal with error without correcting it include preparing the stimulus material to allow for a certain degree of error, frequent recalibrations, and discarding problematic data. Sometimes noise can also be treated by adding more data, e.g. to compensate for low sampling frequency. However, this is not the case for low accuracy, since the additional data will most likely be just as distorted [Holmqvist et al., 2011, 31,32], [Holmqvist et al., 2012, 46]. Therefore this option has little prospect of success.

A careful experimental setup contributes to good data quality, but can only prevent error from certain sources. For example, even when illumination is kept constant and a chin- or forehead-rest is used, errors occur due to other reasons [Drewes et al., 2012, 209], [Lohmeier, 2015, 36], [Martinez-Gomez et al., 2012, 259,260], [Nüssli, 2011, 67]. Likewise, recruiting only participants with optimal eye physiologies and without visual aids [Holmqvist et al., 2011, 141] is not sufficient and would exclude otherwise ideal participants.

5.3.1.1 Stimulus preparation

When constructing stimulus materials, AOIs can be made sufficiently large or surrounded with a generous margin, so the gaze is still registered on the correct AOI despite a certain offset. While this is an adequate approach, it is often not feasible, e.g. because the AOIs are too close together to add extra margins [Feit et al., 2017, 1120], [Holmqvist et al., 2011, 224,225]. Text in general and source code specifically have very small AOIs, which are also located very near to each other. Some source code elements, like operators and separators, even consist of only one or two characters. Increasing font size and adding blank space between lines or words to improve error tolerance is only possible to a certain degree, since it only allows very short texts, which poses a threat to validity of the stimulus material. Furthermore, such an unnatural formatting will most likely influence the participant's reading behavior [Hyrskykari, 2006, 662], [Špakov et al., 2018, 8,9], [Yamaya et al., 2017, 100,101]. The empty spaces can cause more and larger eye movements, less text can be perceived with a fixation, and participants will be irritated by the unusual design.

The stimuli in the EMCR study include extra margins, where they are unlikely to affect the visual behavior. Yet, this preparation is not sufficient to circumvent error.

5.3.1.2 Recalibration

Data quality is usually best right after calibration and can deteriorate over time. Hence, recalibrating during the experiment can help retaining good data quality [Holmqvist et al., 2011, 42], [Hornof & Halverson, 2002, 593,597], [John et al., 2012, 297], [Nyström et al., 2012, 285,286]. Often the operator decides, when to recalibrate, but there are also approaches to automate the process [Hornof & Halverson, 2002].

Assessing the current data quality can be done with required fixation locations (RFL). They indicate the location a person supposedly directed their gaze to at a specific moment. *Explicit RFLs* are targets that the participant was directly asked to fixate, e.g. during calibration validation. *Implicit RFLs* are objects, which are most likely looked at, even though the participant was

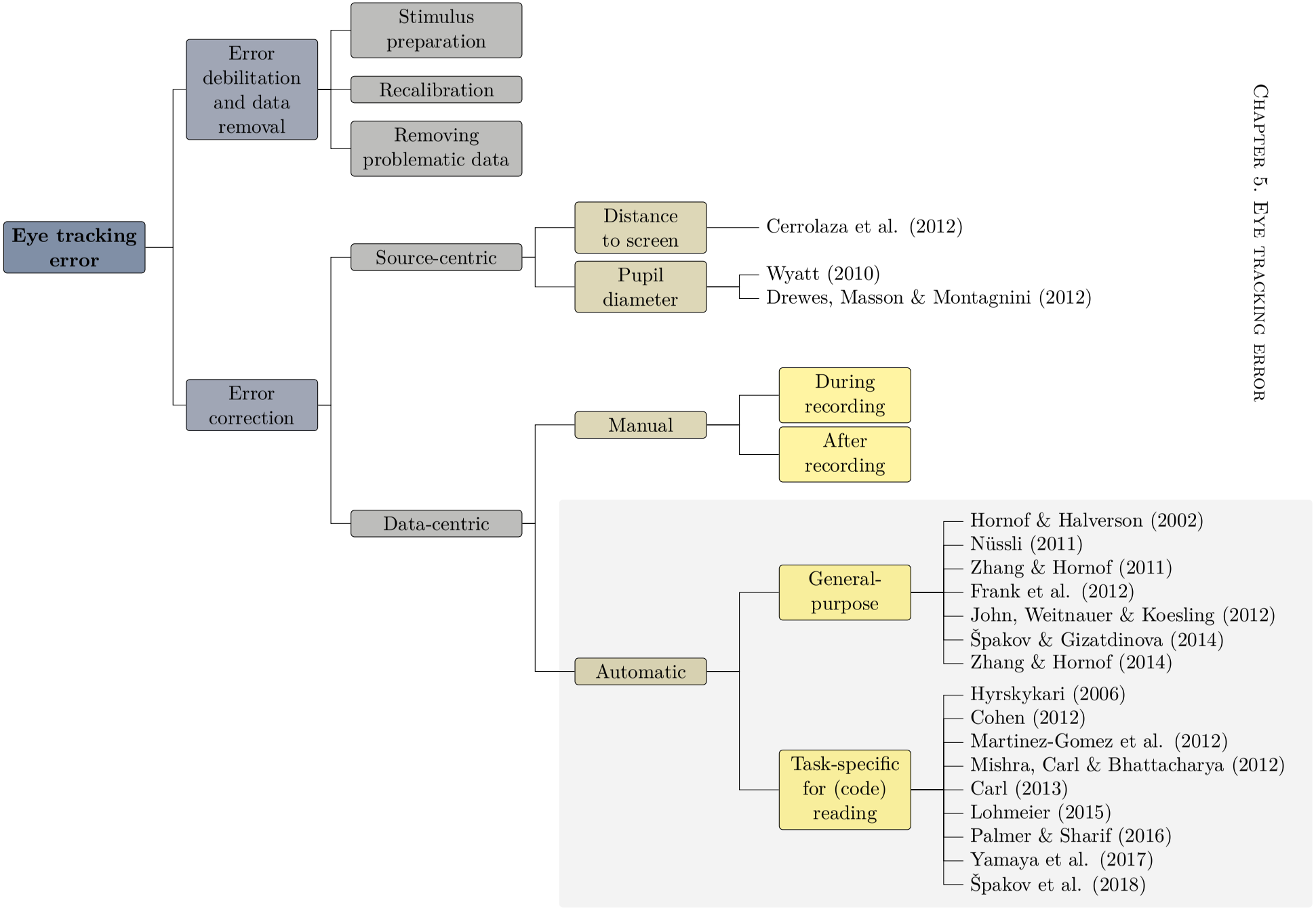


Figure 5.3: Addressing error in eye tracking data. The approaches for automatic error correction (shaded area) are of special interest, as such a step is needed for preparing the EMCR data for analysis.

not instructed to. For example, while clicking a button, the button is usually directly fixated. The disparity between recorded fixation location and RFL can be used to monitor quality and prompt a new calibration, when it falls below a certain threshold [Hornof & Halverson, 2002, 593,594], [Zhang & Hornof, 2011, 835], [Zhang & Hornof, 2014, 95]. The advantage of implicit RFLs is that quality control is integrated smoothly into the recording and does not prolong the experiment duration. However, many study designs, e.g. in reading, do not contain targets that can serve as implicit RFLs [Mishra et al., 2012, 72], [Yamaya et al., 2015, 651], [Zhang & Hornof, 2011, 835]. Hence in the EMCR study RFLs for quality validation had to be called explicitly.

A vital point for recalibration is the timing. Interrupting the recording to calibrate according to current quality is not always possible, since it disturbs the task at hand and consequently the behavior that is studied. However, when calibrating only between stimuli, offsets can still increase within a trial [Hornof & Halverson, 2002, 597], [John et al., 2012, 297], [Mishra et al., 2012, 72], [Zhang & Hornof, 2011, 834,835]. This is the case in the EMCR study, which only allows calibration between texts. Even if it was registered that data quality becomes too low during a trial, the recording could not be interrupted. Considering that head movements were allowed, quite some error could accumulate before recalibration is possible. Sometimes, trials with low data quality can be repeated after recalibration [Hornof & Halverson, 2002, 595], [John et al., 2012, 297]. This procedure might substantially increase the experiment duration, though. Since, participants cannot redo their comprehension process, this approach could not be employed in the EMCR study. If a stimulus text was presented again, participants would show a different behavior, since they had already read and processed at least part of the text.

In the EMCR study, data quality was frequently validated and the system was usually recalibrated several times during the experiment. However, in order to avoid an unreasonably long experiment duration, it was not possible to recalibrate after every single text. Hence, frequent recalibration was employed, but did not prevent error sufficiently.

5.3.1.3 Removing problematic data

The problem of noisy data can also be addressed by discarding problematic trials. However, removing lots of data can unbalance the study design and threaten the validity of results. Especially when participants are able to move their head, almost all trials will contain error. Removing them leaves hardly anything for analysis [Carl, 2013, 1], [Holmqvist et al., 2011, 140,141,224,225], [Holmqvist et al., 2012, 45], [Hornof & Halverson, 2002, 593], [John et al., 2012, 297], while repeating these trials can have unwanted learning effects and prolongs the experiment [Hornof & Halverson, 2002, 595], [John et al., 2012, 297].

5.3.2 Error correction

Optimizing the recording system and environment, as well as the stimuli, can only debilitate error to a certain degree, since e.g. some physiological characteristics remain a challenge. Consequently, often there is no other option than to correct the data. However, while correction is necessary before applying AOI-dependent measures and helps the validity of results, there remains a chance of producing false results.

> "[I]t is important to remember that such shifts of data or AOIs should be made only if it is *obvious* from scanpath visualizations how the repair should be made. For some stimuli, like text or newspaper reading, the scanpaths are so systematic in their alignment to the stimulus that any offset is immediately visible, and its size and direction easily calculable. For general scene images with a varying content, it is often much more difficult to correctly estimate the needed offset repair". [Holmqvist et al., 2011, 225]

Hyrskykari [2006, 670] also points out that reading is one of the tasks that allow quite well to determine the true fixation location. As can be seen in figure 5.3, there are a number of approaches to correct error and correction is widely applied. Generally, an automatic correction method is preferable, since it is objective and reproducible. Zhang & Hornof [2014, 95] categorize error correction methods into source-centric and data-centric. *Source-centric* approaches only compensate errors from a particular origin, e.g. pupil dilation. *Data-centric* methods on the other hand correct errors based on given data, regardless of what caused them. These two types of error correction will be explained in the following sections. Correction methods that require modifications of the recording hardware [Drewes et al., 2012, 211] are out of scope of this work and will not be considered further.

5.3.2.1 Source-centric methods

Source-centric methods correct specific errors, e.g. those caused by changes in the eye-to-screen-distance or in pupil diameter [Zhang & Hornof, 2014, 95].

Source: Eye-to-screen distance
The data quality of many eye tracking systems deteriorates, when the participant moves away from the position he or she was calibrated in. Cerrolaza et al. [2012] demonstrate the negative effects of changing the distance between eyes and screen and present two compensation methods, which each reduces the error by more than half. In the first approach, the participant is calibrated several times at different distances to the screen and the gaze estimation algorithm is adapted to incorporate the varying distances. This considerably reduced the error, but did not eliminate it completely. Also, the additional calibration steps prolong the experiment. Alternatively, the error can be approximated for a particular setup, since it mostly depends on the specific eye tracking device and not on the individual participant. Thus, a pilot group of participants is recorded and the obtained coefficients are averaged for later use in gaze estimation. This approach leads to comparable results as the additional calibration, with the advantage of not lengthening the experiment duration.

Source: Pupil diameter
Pupil size is influenced by a number of factors like luminance and mental workload. When the center of the pupil is used to calculate the gaze location, changes in pupil diameter can be mistaken for eye movements, reaching amplitudes of over 1° of visual angle [Drewes et al., 2012, 209,210], [Holmqvist et al., 2011, 43,391-394], [Holmqvist et al., 2012, 47], [Nyström et al., 2012, 285], [Wyatt, 2010, 1982,1985-1987]. Wyatt [2010] and Drewes et al. [2012] suggest to compensate for error from changes in pupil diameter by collecting additional data with the pupil in a dilated and a constricted state. Wyatt [2010] uses a trial with light and dark condition to establish a function that relates the distance between pupil center and corneal reflex to the pupil size, assuming that for a given diameter the pupil center is always at the same position. Drewes et al. [2012] use the current diameter to weight a bright and a dark calibration. For example, if a pupil diameter is in the middle between the dilated and constricted reference state, both calibrations are factored in evenly. This approach allows corrections both during and after the recording. Both compensation methods reduce, but do not remove error caused by changes in pupil diameter. Their performance depends on the specific participant and the amount of error. Furthermore, since both require to measure the pupil in two reference states, the experiment duration becomes longer. Drewes et al. [2012, 211] also point out, that over- and under-correction occur between the two extremes. Since pupil diameter often does not reach the same amplitudes during recording as during calibration, a third calibration step with a medium bright condition is suggested. However, this would prolong the calibration phase even further.

Source-centric methods can reduce error in gaze data considerably, but do not remove it sufficiently. Offsets between actual and reported gaze location are caused by a combination of errors from different sources and for many there are no compensations methods yet. Even addressing only some major sources, requires several additional countermeasures. Including more steps into the experiment prolongs its duration and makes the recording situation less ecologically valid. This is especially problematic for the EMCR study with novice programmers, since they are recorded repeatedly. Moreover, many such correction techniques have to be set up prior to the recording and cannot be applied to data that was already collected [Palmer & Sharif, 2016, 66]. A more holistic approach is needed, that finds the most likely gaze location.

5.3.2.2 Data-centric methods

In data-centric approaches the error is retrieved based on the data and then removed [Zhang & Hornof, 2014, 95].

5.3.2.2.1 Manual correction

Gaze data is often corrected manually, which can be done during or after the recording.

During recording

Study participant can be provided with information about the currently recorded gaze location and give feedback about its correctness. An example is iDict, an aid for reading foreign language text. When the reader looks at a word for a short while, iDict automatically shows a translation. If the recording is not accurate and iDict therefore translates the wrong word, readers can use the arrow keys to correct the gaze location themselves [Hyrskykari, 2006, 659,662,665,667,668].

This is not possible in the EMCR study, as there is no interaction with the system during reading and the programmer does not receive any feedback about the current quality of the recording. Doing so would create a completely invalid situation, disturb the comprehension process, and distract the attention away from the task at hand.

Post recording

When correcting data after the recording, the data is usually moved to the supposedly correct part of the stimulus. However, it is also possible to place the AOIs over the corresponding data rather than the respective part of the stimulus [Holmqvist et al., 2011, 225]. It is widely acknowledged that manual data correction is very time-consuming and subjective, and not necessarily reproducible [Carl, 2013, 8], [Cohen, 2012, 679,683], [Hyrskykari, 2006, 668], [John et al., 2012, 297], [Mishra et al., 2012, 71], [Yamaya et al., 2016, 37], [Yamaya et al., 2017, 100]. In order to reduce subjectiveness, data can be corrected by more than just one person, e.g. Busjahn et al. [2015a, 260] (see also 5.4.1.1 *Comparison with manual correction*). Unfortunately, this makes the correction process even more work-intensive.

There are also tools to help with manual data correction, e.g. EyeDoctor[1] and FixFix [Topić et al., 2016]. The later especially supports the correction of reading data by providing suggestions for the correct location based on an alignment between gaze segments and the lines of the text. When using the guidance tool, the time needed for correction decreased by 9.7%, while the accuracy slightly increased from 86.4% to 91.2% [Yamaya et al., 2017, 104,105]. However, in order to avoid any bias, an automatic correction method is generally preferable. Moreover, correction is usually carried out on fixations and not on raw data. If the fixations are recalculated using another algorithm or different parameters, the correction has to be done again. Also, the EMCR study contains over 30,000 fixations, making manual correction utterly impossible.

[1] http://blogs.umass.edu/eyelab/software/, last accessed 12/05/2020

5.3.2.2.2 Automatic correction

Automatic corrections have the advantage of being objective and reproducible. Below a number of approaches are outlined and assessed for their suitability for the EMCR study.

5.3.2.2.2.1 General-purpose approaches

Hornof & Halverson [2002]

Approach: Hornof & Halverson [2002] use RFLs (see 5.3.1.2 *Recalibration*) to find and correct systematic error after the recording. The distances and directions from recorded fixations locations to corresponding RFLs provide so-called error signatures across the screen. For each fixation, the four nearest surrounding error signatures are weighted and used for correction. While this method is demonstrated with implicit RFLs, it can be carried out with explicit RFLs as well, as long as they are adequately distributed over the screen.

Evaluation: A scanpath visualization is used to demonstrate that the corrected fixations were on more plausible positions than the uncorrected ones. Additionally, using the RFLs as reference, the average deviation between fixations and their targets was 41 px before correction. After correction, an average standard deviation of the error signature of 12.6 remained.

Limitations: Implicit RFLs cannot be integrated into the EMCR study. Using the points from calibration and validation as explicit RFLs neither provides enough reference points across the screen and nor does it capture error that builds up during the recording, even if calibration or validation was run before every stimulus. Besides, these additional procedures would prolong the experiment even further and decrease the ease of the recording situation. Similar to frequent recalibration, using RFLs for correction is not sufficient.

Nüssli [2011, 52-67]

Approach: Under the premise that gaze mostly lands on AOIs, Nüssli [2011] suggests two correction methods for systematic error. A fitness function determines the fixation likelihood for each point on the screen. The simplest option is to assign '1' to points inside (or near) an AOI and '0' to points outside. More elaborate fitness functions can take the distance to an AOI into account. The *brute force technique* computes the average fitness for every possible offset in a certain range to find one that maximizes the average fitness over all fixations. This method is computationally intensive, but finds the optimal offset. The *analytical technique* on the other hand is a probabilistic optimization, which only approximates the best offset, but allows real-time correction. The stimulus is broken down into simple geometric objects, which can be modeled as probability density functions. The offset, that maximizes the probability of having the measured fixations given the stimulus, is determined through a maximum likelihood estimation. For the computation, it is required that the specific offset per trial remains constant over location and time. Nüssli acknowledges that this is usually not the case and suggests to apply the correction to time- or location-based subsets of the data, e.g. first and second half of the recording.

Evaluation: The brute force technique is evaluated by examining the portion of fixations that land on AOIs. Overall, correction let only to a small improvement of less than 10%, but was much better for some of the trials. However, it is very likely that before correction many fixations were on the wrong AOI, but moved to the correct one. Since there is no ground truth, it is not possible to gauge this improvement. When applying the correction to random gaze, no clear fitness peak appears and the offset variance does not decrease as much as it does for actual data, indicating that the fitness value is not an artifact of the stimulus. Additionally, the distribution of all offsets from all participants was inspected. The tested eye tracker generally recorded the data above the actual location and had a larger vertical than horizontal error. To asses the

analytical method, it was compared to the brute force technique. They had very similar results.

Limitations: In general, this approach is very promising for correcting the EMCR data and even has been used in a study with Java code as stimulus. It will therefore be considered further in 6 *Error correction*. However, both correction techniques compute only a single offset for the complete trial. Applying the correction on time- or location-based subsets of data as suggested is possible, but not entirely unproblematic. When used discretely on each group, fixations in neighboring sections might be shifted unreasonably in relation to each other, especially when only few data points are available for optimization. Hence a more sophisticated technique than simply splitting fixations into subsets is needed.

Zhang & Hornof [2011]

Approach: Zhang & Hornof [2011] propose to correct systematic error by using the nearest object as reference location. When the disparities between fixations and their nearest objects are visualized with a scatter plot, usually a cluster emerges, which approximates the error. The center of this cluster indicates the error's extent and direction and can be roughly estimated by looking at the plot. In order to have a more accurate and efficient way of finding the error, the annealed mean shift algorithm is adapted to eye tracking data for calculating the global mode of the disparities automatically. Finally, fixations are corrected according to the obtained error vector.

Evaluation: Uncorrected and corrected data were visualized as scanpath on the stimulus and the corrected fixations were found to be more plausible. Moreover, a number of correct fixation-to-object mappings were identified by using RFLs, which served as ground truth. Before correction, vertical error reached amplitudes of $-1°$ to $-2°$ of visual angle, while horizontal error was rather small. Both types of error were reduced to almost $0°$ after the correction. The uncorrected data was already quite accurate and 97% of the fixations were correctly mapped to the nearest AOI. After correction the percentage rose to 99.4%. However, the extent of error varied a lot between trials and for some the correction performed considerably better.

Limitations: In order to allow for clustering, several requirements need to be met. For example, targets have to be scattered across the display. If they are arranged regularly, the data might be systematically assigned to wrong AOIs. Unfortunately, text is aligned in a grid-like manner [Lohmeier, 2015, 35], [Mishra et al., 2012, 72]. Furthermore, the method cannot be applied reliably, when several targets can be perceived with a single fixation. However, this is the case in the EMCR study. Especially source code contains many elements with just one or two characters, hence several AOIs can be taken in simultaneously. Moreover, targets should not be too close to each other. Yet, neighboring text elements have only a small blank space between them. The approach also assumes a constant error for the whole trial. To account for error that changes across the screen or over time, it is possible to use region- or time-based visualizations to identify points where the error changes and divide fixations into sub-groups accordingly. Each group is then corrected individually. However, these plots have to be inspected manually, so the correction becomes time-consuming and slightly more subjective. This could partially be addressed by using pre-defined points, e.g. always splitting the trial in half, but that will only work if there are no sudden changes by head movements or other events. Also, some groups might contain too few fixations to reliably form a cluster. Another problem with using sub-groups is, that the correction is used discretely on each group, while the error might change continuously.

Frank et al. [2012, 360-363]

Approach: Similar to the approach by Hornof & Halverson [2002], RFLs are employed in order to measure the quality of the calibration as well as to correct the data. A verification stimulus is included at the beginning, midpoint, and end of the experiment. For the correction, parallel robust regression is used to determine the best translation, and expansion or contraction to match the gaze data to the calibration points.

Evaluation: The corrected data was inspected manually. Trials were only used for analysis, if they contained at least two reference points with sufficient data, for which the corrected gaze locations overlapped with the stimulus at least partly.

Limitations: The participant has to look exactly at the verification stimulus, otherwise the correction might change the data for the worse. As discussed before for Hornof & Halverson [2002], this approach is not applicable to the EMCR data. A verification stimulus would have to include lots of RFLs to cover the screen well and it has to be shown before or after every text, which unreasonably prolongs the experiment, makes the recording situation less ecologically valid, and does not help against error building up during the recording.

John et al. [2012]

Approach: This correction method is primarily intended for correcting systematic error in recordings from head-mounted eye trackers, but it can be used on data from remote devices just as well. It is based on the observation that when a target is looked at several times, the true fixation locations are clustered around that target, but due to distortions the data becomes less clustered. The entropy measure from information theory is used to determine how clustered a group of fixations is. A correction function minimizes the entropy-based error function using recorded fixation locations. The simplex downhill algorithm is used to optimize the parameters of the correction function, though other minimization algorithms are possible as well.

Evaluation: The approach was tested on real and artificial data. When looking at a target, participants pressed a button, so the recorded fixation location can be compared to the target position. The correction consistently increased the quality of the data. However, the extent of the improvement is not stated. Furthermore, the correction was evaluated on a large set of artificial data and compared to the results of correction by linear regression. When applied to undistorted data, the entropy-based method hardly modified the fixation positions, while the linear trend estimation changed the data for the worse. For the distorted data, the entropy-based approach also outperformed the linear regression in all tested variants and removed most of the error.

Limitations: In order to calculate the entropy, targets have to be looked at several times. Even though only a very small number of fixations per target is needed, this requirement is generally not met for (source code) reading. All in all, this approach performs well on stimuli with few AOIs, which are well distributed over the screen. It is not adequate, when many AOIs are not looked at or only looked at once [Lohmeier, 2015, 35].

Špakov & Gizatdinova [2014]

Approach: Špakov & Gizatdinova [2014] propose a method for correcting error in gaze pointing using implicit RFLs. While this correction was developed with real-time application in mind, the general technique can also be used for correcting data after the recording. When the gaze G_{RFL} activates a target, the distance between gaze location and the target's center, which serves as RFL, is stored into a database. Eventually, the database contains multiple entries per RFL. For correction, a weighted mean of the stored distances is added to the gaze location. The closer the uncorrected gaze point is to G_{RFL}, the higher the weight of the associated distance. Since this method only applies to very small objects, a probabilistic approach is introduced for targets of any size. It is assumed that any point inside a rectangular target is equally likely to be looked at. A "shaded" target with the same size as the actual one is placed relatively to the uncorrected gaze point in the same way as the real target is located to G_{RFL}. Given that the intersection rectangle between both targets is not empty, the probability of the gaze point falling into the intersection is calculated. Whether the gaze point is actually mapped to the target, depends on the target's size and the probabilities of neighboring targets.

Evaluation: Using the targets as reference locations, the proposed method is compared to a naive gaze-to-object mapping, the mean target hit rate of 42% improved by 15.7%. Performance was best when the accuracy of the recording was low.

Limitations: Just as with the other approaches including implicit RFLs, this technique cannot by applied to the EMCR data, since such locations are not integrable.

Zhang & Hornof [2014]

Approach: Zhang & Hornof [2014] present a correction method that addresses error, which changes across the screen, provided that the error patterns can be modeled by quadratic equations. RFLs serve as true fixation locations and robust linear regression on the error vectors between recorded gaze and RFL is used to find the parameters of the quadratic recalibration equation. If there are not enough RFLs to cover the screen sufficiently, probable fixation locations (PFL) are added for parameter fitting. Like RFLs, PFLs are targets which have to be looked at in order to successfully complete the trial. However, it is not known, when they are fixated. As already employed by Zhang & Hornof [2011], the nearest targets are used as PFLs. Since the RFL error vectors are more reliable than the PFL ones, they have a higher weight in the calculation.

Evaluation: Using RFLs as reference, the average error decreased from 0.74° to 0.45° visual angle, the medians of the error approached zero and the error range became smaller. Furthermore, visualizing uncorrected and corrected data as scanpaths showed that the correction moved the gaze to more plausible locations.

Limitations: As discussed before, neither RFLs nor PFLs can be included in the EMCR study, i.a. since there is no single text element, which is definitively fixated. Simply using the nearest element as reference location will very likely result in systematically shifting fixations to the wrong target.

5.3.2.2.2.2 Task-specific approaches for (code) reading

Several automatic error correction methods were directly developed for correcting data from reading experiments.[2]

Hyrskykari [2006]

Approach: This correction method was developed with the reading aid iDict (see 5.3.2.2.1 *Manual correction*) in mind. It allows to automatically correct systematic and variable vertical error. As long as the reading seems to continue along a line, the *sticky lines* technique assigns even heavily ascending or descending gaze to the currently active line. The *magnetic lines* routine detects the sweep from the end of a line to a new one. The new line serves as RFL, so the correction does not only affect the current line, but also the surrounding ones. If these two automatic techniques are not sufficient, it is possible to make manual corrections during the recording, which also provide RFLs for further automatic correction.

Evaluation: The magnetic and sticky line techniques were assessed using texts of varying line spacing. For evaluation, no additional manual corrections were carried out during recording. The correct lines were established manually afterwards. For all conditions, the portion of fixations that were assigned to the correct line were higher after correction than before, but the extent of the improvement depended on the spacing. In the best condition with 1.5 spacing, the percentage of correctly assigned fixations increased from 56% to 86%. The performance can be considerably increased by allowing manual correction during the recording.

Limitations: While this correction is intended for real-time application, the sticky and magnetic lines techniques can be applied after the recording just as well, potentially even combined with manual post-recording corrections. However, the approach is based on a linear reading approach, which does not apply to source code reading, which includes other behaviors such as skimming [Lohmeier, 2015, 35]. While the approach allows for a certain amount of regressions, it does not perform well, when the reading is not fluent and the gaze switches lines often, going back and forth, which is common in code reading. In case the gaze is mapped incorrectly, it can remain wrong for quite a number of fixations, making this approach unsuitable for code reading.

Cohen [2012]

Approach: Cohen's correction approach uses linear regression to assign fixations to a line in the text. Each text line has an associated regression line. Based on the assumption that the error is largely constant over the screen, all regression lines have the same slope, vertical offset, and standard deviation. At first, the regression lines run through the start of the text lines. If fixations are recorded above or below the text lines, the vertical offset is used to shift the start of the regression lines accordingly. After calculating the set of parameters that maximizes the likelihood of the fixations, they are mapped to the line of text with the highest-likelihood regression line.

Evaluation: For assessment, the automatic corrections were compared to manual ones. Overall, the human and the software agreed on 99.78% of the total number of fixations.

Limitations: This approach corrects only vertical error and is based on the assumption that the offset remains relatively constant over time and across the screen, which is not the case for the EMCR data. Moreover, the many short lines in source code might make it difficult to fit the regression lines.

[2] The technique by Mazzei et al. [2014], will not be considered here, since it applies to reading and note taking on paper, and is based on a dual-camera setup with a head-mounted eye tracker.

Martinez-Gomez et al. [2012]

Approach: This correction method uses feature-based image registration, which matches two images by spatial transformation of the source to the target image. It transforms the image representation of gaze data to align to the image representation of the stimulus text without assuming a specific reading behavior. In order to limit the number of parameters needed for transformation, only vertical error is considered. Ideally, every gaze sample can be mapped to a word. Pixels in the gaze image get a low score, if a gaze sample occurs on them. Likewise, pixels in the text image have a low value, if there is a word at their location. The sum of the absolute differences of pixels between the two images is used as measure of alignment. In order to find parameters that minimize the alignment, Monte Carlo sampling, multi-resolution optimization, and multi-blur optimization are employed. While this technique was intended for reading in a broad sense, it can probably be applied to other domains as well, as long as the stimulus includes enough distinct features for transformation.

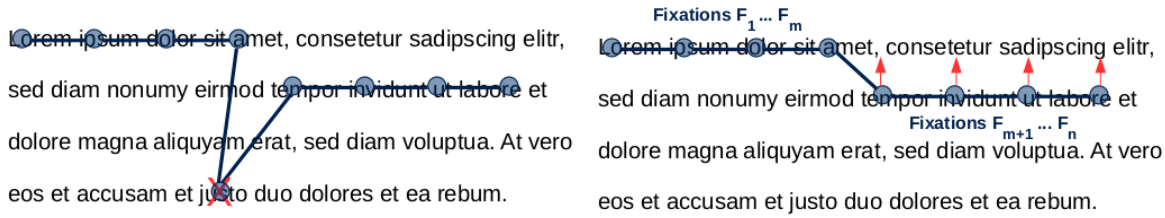
Evaluation: The approach was tested using linear reading and skimming. The corrected linear reading data is mapped to words and compared to the sequence of words in the text using the Levenshtein distance. For skimming, the F-score is calculated, the weighted average between precision and recall of shown and mapped words. A naive error correction without mapping serves as baseline for comparison. Multi-blur optimization performed best and significantly better than the baseline. The results were generally better for linear reading than for skimming. The reading data was also corrected manually. The best automatic correction has an accuracy of ca. 70% relative to the manual version. Skimming data was not compared to a manual correction.

Limitations: This approach addresses only vertical error. Horizontal correction could be added, but will make the optimization very complex. While the general idea is very intriguing, since it is applicable for varying reading behaviors, it only considers error that is largely constant across the screen. Additionally, it does not perform well enough on word level, which is a necessity for the EMCR data.

Mishra et al. [2012]

Approach: Similar to Hyrskykari [2006], this approach is based on the assumption that reading is mostly carried out linearly from left to right and line after line. The underlying heuristics were derived by inspecting a large number of recordings from translation, post-editing, and reading experiments. Correction can be applied during or after the recording, but only addresses vertical error. First, fixations are moved to the closest line. Then, transient fixations are discarded. These are short fixations that occur between two other fixations, which are close to each other, but far away from the short fixation (figure 5.4a). Given a good calibration and enough line spacing, the first fixations on a line are assumed to be correct. If they are mapped to the same line, it can be reasonably assumed that successive fixations further to the right belong to that line as well, even when they are recorded somewhat above or below the current line (figure 5.4b), since it is unlikely that the gaze jumps somewhere else, while in the middle of a line. The amount of fixations m which indicate the correct line is determined by manually inspecting sample recordings. If the first m fixations are located on different lines, the most probable one is found by a ranking based on the fixation frequency distribution among the lines and fixation durations.

Evaluation: A qualitative analysis was done by visually inspecting replays of uncorrected and corrected recordings. After correction, the data was found to be more plausible and less noisy. Furthermore, the automatic corrections were compared to manual ones. While the automatic approach changed almost all data, only fixations with huge offsets were corrected manually.



(a) Removing a transient fixation from the scanpath.

(b) Moving fixations $F_{m+1}...F_n$ to the same line as the first m fixations.

Figure 5.4: Correction steps concerning transient fixations and moving fixations to the same line as their predecessors

Therefore, it was tested, which portion of the manual corrections were also carried out by the automatic approach. Depending on the chosen number of first fixations m, the agreement between automatic and manual correction ranged between 55% and 81%. However, this evaluation method is somewhat problematic, since only extreme cases are captured and there is no information on how the correction performed on the majority of fixations.

Limitations: The premise that reading is mainly carried out line after line, from left to right, makes this method unsuitable for code reading. Additionally, the success of the correction depends on the first m fixations being fairly exact. If they are amiss, so will be all following fixations for that line. In order to ensure that the correct line was detected, additional information is needed, e.g. in form of RFLs, which are not available for the EMCR data. Finally, transient fixations cannot necessarily be removed in code comprehension studies, since it is very difficult to decide, whether these fixations are noise or serve a purpose.

Carl [2013]

Approach: This correction approach is task-specific for translation, where a foreign-language text is presented in one window and its translation is typed into a second one. It is included here, since it addresses reading as part of translating. Background knowledge about translating allows to build correction heuristics, e.g. the gaze jumps between words currently being translated and their translation and consecutive fixations are very likely to fall on neighboring words. Fixations are computed from raw data and assigned to the nearest character (naive mapping), which often lies in the line above or below the one actually looked at. This first mapping is extended into a lattice of possibly looked-at symbols. Additional fixations are calculated using left-only, right-only, and left-right average samples to utilize the supposedly more precise data from the dominant eye. However, as eye dominance is unknown, all options are tested and the characters closest to these fixations are added as candidates to the lattice. Furthermore, the characters in the line above the uppermost, and below the lowest mapping are included to account for systematic error. A dynamic programming algorithm calculates the optimal path through these possible mappings, using the distance between consecutive fixations, and the window the fixations occur in. If fixations appear in the same window, it is assumed that the translator reads some text and the gaze progresses forward. If the window changes, the gaze probably moves near the translation of a text that was previously read. Additionally, the best mapping option is expected to remain stable for some time.

Evaluation: Manual corrections by three people were compared to each other, to a naive, and to the presented re-mapping approach. All versions agree for over 97% of the fixations on the window the translator was looking at. The two windows are rather large AOIs, though. The biggest consensus was reached between the humans. Comparing manual corrections with

the naive mapping produced a slightly better result than with the re-mapping. However, the differences are minimal. With regard to the average character distance between the corrected versions (see 5.4.1.1 *Comparison with manual correction*), the manual versions are quite close to each other (distances between 8.6 and 12.6), the naive and the re-mapping approach much less so (distances between 34.8 and 37.7, and 32.8 and 35.9). The extent of the distances depends on which window the fixation occurred in. Especially for fixations that are mapped to the same window, the performance of the re-mapping is better than of the naive method. Larger distances are mostly the result of mapping a fixation to different windows, e.g. last line of the upper window and first line of the lower one. Moreover, the corrected versions are inspected to see how well they comply with a cognitive model of translation. After re-mapping, the data agrees better with the expected behavior.

Limitations: This approach also addresses reading, but in a very different context. The presented heuristics strongly depend on the two-window setup and on writing new text during the recording, none of which is part of the EMCR study.

Lohmeier [2015, 35-44]

Approach: This error correction method was especially developed for program comprehension. It seizes the irregular layout of source code with indentions and lines of varying lengths. For every fixation several offsets and linear factors are tested to find a set, which results in a minimal average distance between fixation locations and targets. The respective targets are determined by using an asymmetric rectangular fixation assignment window around the measured gaze position, which corresponds to the size of perceptual and word identification spans in reading. During Lohmeier's experiment, participants worked with a specific IDE. Finding the intended target is attuned to this stimulus and includes identifying the user interface control at the corrected fixation position, and inside the control the closest word.

Evaluation: The error gradients from the parameter test, as well as the corrected data were plotted and inspected visually for plausibility. Furthermore, corrected and uncorrected data was compared with descriptive statistics. The automatic correction reduced the average error by about a third.

Limitations: This approach for gaze data in program comprehension contains several very promising elements for correcting the EMCR data and will be further evaluated in 6 *Error correction*. Horizontal error was corrected manually, but can be included in the computation without further ado. However, the stimulus source code was shown in a specific IDE, whose elements help to identify fixation targets. A different approach has to be found for this step, since the EMCR stimuli are natural-language texts and source codes on an otherwise blank screen. Furthermore, the error has to remain largely constant across the screen and over time. Similar to the approach by Nüssli [2011, 52-67], this technique can be applied to subsets of data, e.g. from different screen regions, even though such a step needs a more elaborate implementation than merely dividing fixations into subgroups.

Palmer & Sharif [2016, 65-68]

Approach: This approach was also developed for gaze data from program comprehension, but is potentially also applicable to reading or even other stimuli that meet the underlying prerequisites. First, clusters of temporally dense fixations are formed using a lookback queue. Then, all clusters receive a score depending on how many of their fixations fall into an AOI. Clusters whose score falls below a threshold are corrected using a hill-climbing algorithm. In case no fixation from the cluster is inside an AOI, the cluster is moved towards the closest AOI. During hill-climbing, scores are determined for the cluster's current position and its four

neighbors. If any of the neighbors score higher than the current position, the cluster is moved in that direction. At the beginning, the cluster moves half its width and height, but every cycle, the distance decreases by half until it reaches one pixel. The hill-climbing repeats until the current position has the highest score.

Evaluation: The automatic correction results were compared to manual ones from two annotators working together. Using code lines as AOIs, both automatic and manual corrections agreed on 89.78% of the fixations, on element level the agreement was 59.47%.

Limitations: This is a very promising approach, because it was developed for code reading and uses a variable offset for correction. However, as a start, it was aimed at correcting data from novices and assumes a somewhat linear reading behavior. Also, so far it only performs well on line-level, not for single code elements. Further adaption is needed in order to also account for non-linear reading during program comprehension.

Yamaya et al. [2017]

Approach: Yamaya et al. [2017] developed a method for correcting vertical error in reading data. Fixations are broken down into sequential reading segments, mainly based on return sweeps. When a text is read linearly from top to bottom without re-reading or skipping, the number of segments is the same as the number of lines. Since the actual reading behavior somewhat differs from this model, reading segments and lines do not match perfectly. Hence, transitions between consecutive segments are classified into different types, which were derived empirically from observing gaze patterns. These types include *progressing*, i.e. left-to-right movements to read the next AOI, and *short re-reading* on the same line, i.e. reading again from the beginning or halfway along the same line. A classifier based on machine learning labels the transitions using features like segment length and saccade angle. Then, segments are either divided or concatenated, depending on the type of transition between them. Finally, dynamic programming is used to find the best alignment between segments and lines. The method builds on previous work by Yamaya et al. [2015] and Yamaya et al. [2016].

Evaluation: The classification was evaluated by comparing transition labels assigned by a human to those of the machine approach. The automatic classification reached an accuracy of 84%. The fixation data was also corrected manually by several annotators, who agreed on 84.4% of the fixation-to-word mappings. Three techniques were compared to the manual set: a naive mapping that assigns fixations to the nearest word, the dynamic programming-based segment-to-line mapping, and segment-to-line mapping with transition classification. The naive approach reached an average accuracy of 69%, segment-to-line mapping 72.3%, and segment-to-line mapping with classification 87%.

Limitations: The method assumes a linear reading behavior. Even though non-linear reading is to some extent taken into account, e.g. by tolerating regressions within a line, it does not perform well when larger parts of the text are re-read or the text is not read completely, which is both very common in source code reading. Moreover, only vertical error is addressed.

Špakov et al. [2018]

Approach: Two methods are introduced based on a linear reading model, i.e. lines are read from left to right and rarely skipped, after reaching the end of a line, the gaze moves to the beginning of the next line, most words are looked at, and ca. 10-15% of the saccades are regressive. This is similar to the premises in by Hyrskykari [2006], Mishra et al. [2012] and Yamaya et al. [2017]. *Immediate Mapping* is carried out during the recording, using the screen locations of lines and words, and the mappings of previous fixations. It has two components. In relative mapping the shift relative to the previously mapped fixation serves as indicator for the correct line. A

weighted average of the distances between mapped fixations and their respective lines serves as additional verification. If no relative mapping can be established, absolute mapping starts by calculating the vertical distance from the fixation to each line. If only one line is within the specified distance, the fixation is assigned to it. When both the line above and below the fixation are candidates, it is checked, whether one already has fixations mapped to it, while the other has none. If so, the line without fixations is favored, otherwise the closest line is chosen. Subsequently, fixations are mapped to words within the line, usually to the closest one. However, motivated by the asymmetric reading span, certain fixations that occur at the end of a word are assigned to the next word. *Deferred Mapping* corrects the data after the recording. It forms sequences of fixations that are close enough to each other to be made when reading the same line. These sequences of progressive reading are mapped to lines. In order to assign fixations to a word within the line, the horizontal coordinates are scaled according to the horizontal extents of the words in the line. Then the fixation is mapped to a word, using the "effective" word width. If the actual word width exceeds a certain threshold or the word already has assigned fixations, this is a fraction of the actual word width. Otherwise, effective and actual width are identical. Finally, transition fixations, which occur during corrective saccades after moving the gaze to the start of a new line, are removed.

Evaluation: The lines detected by both methods are compared to those obtained manually by two human judges individually. For trials with complete agreement between manual mappings, the median agreement between immediate and manual mapping is 92.9%, for deferred and manual mapping it is 96.2%. In more ambiguous trials with 95% or less agreement between judges, 88.4% of the immediate and 94.4% of the deferred mappings agree with at least one of the manual versions. Thus, deferred mapping performed better that immediate mapping. 17% of the trials were not used for evaluation, e.g. because of large data loss or because the participant was not reading linearly. Fixation-to-word mappings were not assessed, as automatic scaling is assumed to be more accurate than human judgment.

Limitations: Both mapping methods are only applicable when the text is read linearly and without interruptions. Since this prerequisite is not met in code reading, none of them can be used for the EMCR data.

5.3.3 Conclusion

There are various ways to address error in gaze data. In some studies it is possible to prevent error sufficiently or remove problematic data. However, often it is unavoidable to correct the error and various approaches already exist. Yet, most of them cannot be applied to the EMCR data, since they are based upon some kind of required or probable fixation location, which cannot be integrated into the EMCR study. Techniques that were developed especially for data from reading studies mostly assume a rather linear reading behavior and are therefore unsuitable for code reading. However, two approaches were identified that can presumably be adapted for correcting the EMCR data: Nüssli [2011, 52-67] and Lohmeier [2015, 35-44]. These will be explored further in 6 *Error correction*.

5.4 Evaluation approaches for error correction

Evaluating the performance of error correction methods is not entirely straightforward, since it is difficult to obtain a ground truth for comparison [Nüssli, 2011, 60]. Also, the performance often varies strongly between trials and depends on the extent of error, e.g. Nüssli [2011, 61], Špakov & Gizatdinova [2014, 293,294], Zhang & Hornof [2011, 839]. Below, a systematic summarization of the identified evaluation approaches is given. Occasionally, several techniques are used together in order to corroborate their findings, e.g. Carl [2013, 8-11], John et al. [2012, 298-300], Mishra et al. [2012, 73,77,78], Zhang & Hornof [2011, 838-840].

5.4.1 Real data

The majority of assessment procedures use real gaze data. Partly, correction is carried out on realistic tasks, i.e. the same kind that the correction method is intended for, e.g. Cohen [2012, 681,682], Hyrskykari [2006, 668], Nüssli [2011, 60], Palmer & Sharif [2016, 67], Špakov et al. [2018, 14,15], but points or grids were also used [Cerrolaza et al., 2012, 207], [Špakov & Gizatdinova, 2014, 293]. In order to test the robustness of their method, Špakov & Gizatdinova [2014, 293] even created highly noisy data on purpose by prompting participants to change their posture on the chair and to walk a few meters and sit back down to continue the recording without recalibration.

5.4.1.1 Comparison with manual correction

Data can be corrected manually and compared to automatically corrected data. The size of the AOIs used for evaluation varies between characters [Carl, 2013, 9,10], words [Martinez-Gomez et al., 2012, 259,260], [Yamaya et al., 2017, 104], lines [Hyrskykari, 2006, 668], [Špakov et al., 2018, 14-16] or even larger areas of the screen [Carl, 2013, 8-10]. Since manual correction is usually quite time-consuming, often there are only small sets of data for comparison [Hyrskykari, 2006, 668]. While some only used a single person to correct the data [Cohen, 2012, 682], others had two or more correctors working independently. These different versions are tested against each other to ensure a high quality of the correction before using them as ground truth [Carl, 2013, 8,9], [Špakov et al., 2018, 14-16], [Yamaya et al., 2017, 104]. A similar option is to have several people correct the data together [Palmer & Sharif, 2016, 67]. Enrolling more than one corrector and ascertain that the corrections largely agree, before accepting the manually edited data as correct, is a more reliable, improved version of the manual approach and reduces subjectiveness. Mishra et al. [2012, 78] performed manual correction only on fixations with large offsets, then tested which part of this correction is also carried out by the automatic approach. This procedure is problematic, since the automatic correction method modifies most of the fixations and it is unknown how valid those shifts are. Hence the whole data set has to be examined.

Instead of comparing the exact coordinates of the corrected fixations to quantify the correction performance, often the percentage of fixations that were assigned to the same target is used [Carl, 2013, 8,9], [Cohen, 2012, 683], [Hyrskykari, 2006, 668], [Martinez-Gomez et al., 2012, 259,260], [Mishra et al., 2012, 78], [Palmer & Sharif, 2016, 67], [Špakov et al., 2018, 15,16], [Yamaya et al., 2017, 104]. This measure is sometimes called *hit percentage* or *accuracy*. Other measures, that are introduced concern the distance between manually and automatically corrected fixations. The *average character distance* used by Carl [2013, 9,10] reflects the textual offset between two versions of the data, irrespective of the physical distance. A mis-mapping within a line has a smaller distance than a mapping to different lines, even if the physical distance is the same. Building on this measure, Yamaya et al. [2015, 656,657] define the *average*

word distance, i.e. the average normalized distance between the offset positions of two corrected data sets.

5.4.1.2 Reference locations

There are several ways to employ reference locations for evaluating a correction approach. Some stimuli have implicit RFLs or PFLs, e.g. targets in a visual search task [Hornof & Halverson, 2002, 597-603], [Zhang & Hornof, 2011, 839-840], [Zhang & Hornof, 2014, 97,98]. Other experiments use explicit RFLs, e.g. participants are asked to press a button, when they fixate certain points [Cerrolaza et al., 2012, 207,208], [John et al., 2012, 298,299], [Špakov & Gizatdinova, 2014, 293,294]. The disparity between fixation and reference location, and the percentage of correct target assignments can be used to assess the correction's performance.

Martinez-Gomez et al. [2012, 259,260] tested their method with careful reading and skimming. They asked participants to mimic an ideal reading behavior by looking at each word, one after the other without skipping words or making regressions. Understanding the text was not necessary. Thereby the sequence of read words largely corresponds to the sequence of words in the text and a measure based on the Levenshtein distance is used to determine their similarity. However, it is not feasible to read this linearly all the time, so a perfect match between the two sequences is rather impossible. For skimming, only some words were kept visible and participants looked once at each word in whichever order they wanted. The evaluation metric was the F-score, the weighted average between precision and recall of the words that were looked at and the words identified as correct by the correction method. Drewes et al. [2012, 209-211] and Wyatt [2010, 1983] asked participants to look steadily at a central target on the screen and changed the brightness to demonstrate that changes in pupil size are mistakenly recorded as eye movements, even though the gaze remained on the target.

5.4.1.3 Visualizations

Another option is visual inspection of scanpaths and other visualizations in order to determine whether the corrected data is plausible or to assess and illustrate aspects of the correction. Often uncorrected and corrected data or corrected data from different correction techniques are presented together [Cohen, 2012, 681,682], [Hornof & Halverson, 2002, 601-603], [John et al., 2012, 298,299], [Mishra et al., 2012, 73,77,78], [Palmer & Sharif, 2016, 67,68], [Yamaya et al., 2017, 104,105], [Zhang & Hornof, 2011, 838,839], [Zhang & Hornof, 2014, 97,98]. Carl [2013, 10,11] reviews corrected scanpaths to determine, whether they fit with a cognitive model of the translation process. This is very close to an expert inspecting a scanpath for plausibility, except that an explicitly formulated behavior serves as guidance.

Furthermore, the error and its horizontal and vertical components can be plotted before and after correction [Lohmeier, 2015, 41,42], [Nüssli, 2011, 63-65], [Zhang & Hornof, 2014, 97,98]. Lohmeier [2015, 39-41] also visualized the error gradients from brute force parameter testing, as well as the y-offset and linear factor, to ensure that optimal results were found. Moreover, Nüssli [2011, 61-66] plotted fitness values in the offset space. The fitness peak indicates an optimal offset. If no systematic error is found, the peak is centered at zero. Additionally, histograms of the fitness difference between uncorrected and corrected fixations and of the differences between the brute force and the analytical method, as well as the development of the offset variance with the number of tested fixations are depicted. Martinez-Gomez et al. [2012, 259] present the respective solution spaces for the tested optimization techniques.

5.4.1.4 Comparison with another correction method

Several methods are evaluated by comparing their results to those of another approach. Often a simpler, "naive" method serves as baseline, e.g. assigning fixations to the nearest element [Carl, 2013, 8-10], [Martinez-Gomez et al., 2012, 259,260], [Špakov & Gizatdinova, 2014, 293,294], [Yamaya et al., 2017, 104]. Apart from that, several correction methods are introduced and compared to each other [Cerrolaza et al., 2012, 207,208], [Nüssli, 2011, 65,66], or several versions of the same method are tested [Martinez-Gomez et al., 2012, 259,260], [Mishra et al., 2012, 78], [Yamaya et al., 2017, 104]. The general problem with comparing one method to another is that the reference method might also not work correctly. Thus, if both methods come to different results, it is difficult to determined which approach is correct [Chapanis, 1951, 1179-1181].

5.4.1.5 Further measures

Since no ground truth is available for comparison, Nüssli [2011, 61-64] uses the distribution of fitness improvement over all participants to check that after correction more fixations are assigned to AOIs. However, this gives no information about the amount of fixations on the correct AOI, so the improvement cannot be fully quantified. Other evaluation techniques include employing an eye simulator [Cerrolaza et al., 2012, 206,207] or a scleral search coil simultaneously with the eye tracker [Drewes et al., 2012, 209-211], and using descriptive statistics at different stages of correction [Lohmeier, 2015, 40,43].

5.4.2 Artificial data

In order to have an actual ground truth and know the correct fixation locations, John et al. [2012, 299,300] work with artificial data. In doing so, huge amounts of test data can be generated, which is hardly feasible with real participants. A "true" data set is created, in which ten percent of the fixations are replaced by white noise to simulate artifacts. This data is distorted with increasing complexity. Then the proposed correction method is performed on the distorted as well as on the original data to verify that correction does not alter correct fixation locations. To test, how well the correction results agree with the true data, the mean distance between the corrected and the undistorted data is calculated.

Nüssli [2011, 61,63,64] compares the results from automatic correction on real and random gaze. All fixation locations are replaced by random points that are distributed all over the screen. To ensure that the results are not an artifact of the stimulus, the stimulus and the fitness function are identical for both data sets. For the real data, the offset variance decreases as more fixations are added to the computation and the offset becomes more distinct. However, when processing the random positions, no clear fitness peak appears and offset variance decreases much less.

5.4.3 Conclusion

A number of different methods were developed to asses the performance of error correction approaches. The need for some kind of ground truth to base the evaluation on makes the assessment a difficult task. When manually corrected data is used to evaluate the performance of a correction approach, it is never completely certain, that the manually determined fixation location is correct, even if several people agree on it. Also, manual corrections are very time-consuming. Inspecting the corrected data to determine how plausible it is, is also subjective. Using reference locations for evaluation is often not possible or at least not with a realistic stimulus. While using artificial data avoids these problems, it is difficult to judge how realistic the applied distortions are and whether the data can really be seen as a stand-in for the real

data that has be corrected. Therefore, for evaluating the error correction of the EMCR study a combination of several methods will be used, including reference locations, manually corrected data, and artificial data.

5.5 Conclusion

In order to analyze the EMCR data, it is necessary to establish which AOIs are looked at. Due to a number of reasons, there is often an offset between the recorded gaze and the actually fixated location. Reviewing a variety of options to address such errors yielded two useful correction approaches for further inspection, as well as relevant means for evaluation.

6 Error correction

6.1 Introduction

In 5 *Eye tracking error* two correction methods were identified that can serve as basis for developing an automatic correction approach for the EMCR data. Results from different variants of this novel correction are compared to reference locations in form of mouse clicks on the fixated screen location. The methods that perform best for NT and SC respectively are evaluated further using data from the EMCR study, which was corrected manually, as well as artificial data to ensure that neither accurate data is distorted nor meaning created from random data.

6.2 Correction approaches

The approaches by Nüssli [2011, 52-67] and Lohmeier [2015, 35-44] were determined as the most suitable ones for further investigation (see 5.3.2.2.2 *Automatic correction*). The core idea of Nüssli's approach is to maximize the amount of fixations on AOIs, while Lohmeier minimizes the disparities between fixations and their targets. These two approaches are briefly outlined again below, providing some additional details that are relevant for the development of a suitable correction method for the EMCR study data.

6.2.1 Nüssli 2011

Nüssli's approach is based on the assumption that participants mostly look at AOIs, not on blank space and computes an offset, which maximizes the number of fixations on AOIs. While participants might sometimes look at empty or irrelevant areas, or even away from the screen, e.g. when pondering something, they predominantly focus on AOIs as long as the task requires information that has to be obtained visually from the stimulus and not by merely thinking about it. This assumption should hold especially for stimuli like NT and SC, which are only partially covered with elements, and in the case of SC additionally have a very distinct AOI structure.

Several strong indicators support the underlying premise. When testing potential offsets on data from actual studies and plotting the resulting percentages of fixations on AOIs into a two-dimensional landscape, a distinct peak appears, which indicates the optimal offset. For randomly distributed fixations, no such peak forms. Also, as soon as enough fixations are taken into account, the offset variance decreases much more for data from solving an actual task than for random points. Hence there is a higher confidence in the offset computed for real data than for random fixations.

Two implementations of the correction approach are proposed. In the *brute force technique* the so-called fitness function assigns a fixation likelihood to each screen location. A straightforward option is to assign '1' to points inside AOIs and '0' to those outside. It is also possible to devise fitness functions that compute a score between '0' and '1', depending on the distance to an AOI. In order to find the offset which maximizes the average fitness over all fixations, fitness values are computed for all offsets in a given range. This brute force procedure finds an optimal offset, but is computationally expensive, especially for large numbers of possible offsets. Therefore, Nüssli also introduces an *analytical technique*, which approximates the optimum and requires less computations.

The whole approach further relies on the assumption that there is a specific error for a trial or subject and that this error is constant over space and time. Nüssli acknowledges that error may change in both dimensions, but as a start, takes these premises for granted and mentions the option to run the correction on subsets of data. This alternative will be explored more thoroughly in section 6.2.3 *Novel approach*. For further details and an evaluation of Nüssli's approach see 5.3.2.2.2.1 *General-purpose approaches*.

6.2.2 Lohmeier 2015

Lohmeier developed a correction method for an experiment on SC using an IDE. It employs a fixation assignment window, whose size is based on the perceptual and word identification spans in reading. Fixation targets are identified using features of the stimulus IDE, e.g. distinct control elements like buttons, as well as the specific shape of SC with lines of variable length and varying start and end points. Different sets of parameters are tested on the fixation positions to find the one that results in the smallest average disparity between fixations and their targets. These parameters consist of an offset and a linear factor for scaling the fixation coordinates along the axes. Further details and an evaluation of this approach can be found in 5.3.2.2.2.2 *Task-specific approaches for (code) reading*.

6.2.3 Novel approach

Drawing from the approaches by Nüssli and Lohmeier, an enhanced correction method is devised for the EMCR data. Several variants of this correction are evaluated for their suitability.

6.2.3.1 Error function

To get the corrected fixation location f_{corr}, Nüssli applies a constant offset o to the uncorrected fixation coordinates f_{uncorr} (equation 6.1).

$$f_{corr} = f_{uncorr} + o \tag{6.1}$$

Lohmeier has a more refined correction function. Additionally to the offset, it includes a linear factor a that can somewhat compensate for error varying over the screen location (equation 6.2):

$$f_{corr} = a * f_{uncorr} + o \tag{6.2}$$

This approach allows more fine-grained adjustments to changing errors. An example can be seen in figure 6.1. The concept of combining a factor for scaling and an offset for translation is also used by Martinez-Gomez et al. [2012, 258]. Furthermore, equation 6.2 includes equation 6.1, since the linear factor a can be set to '1'. Hence, this more powerful option is used. Nonetheless, in the evaluation results for $a = 1$ will be presented as well for comparison.

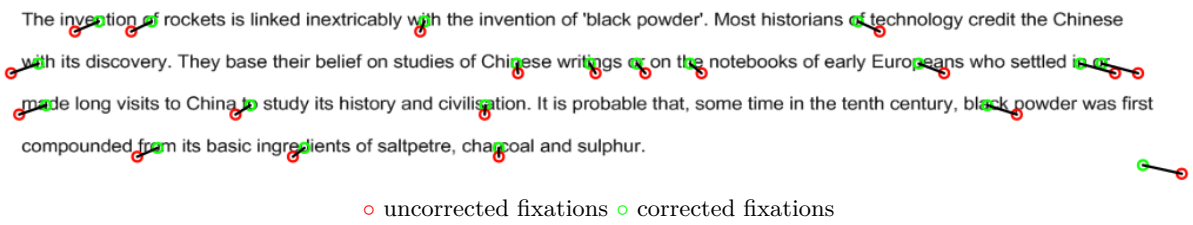


Figure 6.1: Scaling and translation: Applying a linear factor and an offset to the uncorrected fixations allows to correct the rather small error in the middle of the screen as well as the errors at the left and right side of the text, which not only have a larger extent, but also opposite directions. Correcting the data with only an offset cannot compensate this changing error.

6.2.3.2 Finding parameters

Both presented approaches have the underlying assumption that gaze is mostly directed at AOIs. In order to optimize the parameters for correction, Nüssli maximizes the amount of fixations on AOIs, while Lohmeier minimizes the disparities between fixations and AOIs. The EMCR study does not include an IDE, so Lohmeier's approach of identifying fixation targets has to be adapted to stimuli which contain only text. Nüssli's approach on the other hand works with text stimuli without further ado. So the number of fixations on AOIs will be maximized. The brute force variant finds an optimum, but is computationally expensive, while the analytical technique only approximates the optimum, but is much faster. For analyzing the EMCR data, exact results are more important than the time needed for computation. Furthermore, brute force values would have to be computed for at least part of the data in order to set the parameters for the analytical method, so the brute force technique is chosen.

The EMCR text stimuli are rather dense and often include just a few pixels of empty space between AOIs. There is hardly any potential for computing a gradual score depending on the distance to an AOI, as fixations that fall within the text area have a high chance of getting a score of '1' anyway, because they will most likely land on an AOI, even if it is the wrong one. Considering distances to AOIs is mostly relevant for locations near the text margins, where more empty space surrounds the AOI, e.g. above the first line and in SC for lines that are longer than their preceding and subsequent lines. Instead of implementing a distance-based fitness function, which would be applied to all points, AOIs at locations that are less dense will be adapted to accommodate fixations with a somewhat greater error (see 6.2.3.3 *Adapting AOIs*). Hence, the straightforward fitness function (equation 6.3) is adopted. It also allows to interpret the resulting overall fitness value as percentage of fixations on AOIs.

$$fitness(x, y) = \begin{cases} 1 & \text{if there is an AOI at location (x,y)} \\ 0 & \text{else} \end{cases} \tag{6.3}$$

Nüssli [2011, 54] tested offsets between -100 and +100 px. Lohmeier [2015, 40,41] used offsets in a range of -100 to +100 px and -140 to +140 px, and linear factors of [1, 1.01, 1.02, ..., 1.3] and [1, 1.01, 1.02, ..., 1.6]. Despite the computational effort, automatic correction with linear factor and offset will be tested for the EMCR data both in horizontal and vertical dimension in order to find the best possible parameters.
Chosen ranges:

- Linear factor a:
  0.9 to 1.1 with a step size of 0.01
- Offset o:
  -100 to +100 px with an increment of 1 px

These parameter ranges with a very small step size allow to explore the effect of scaling towards the middle and the edges as well as very fine-grained testing. The stimuli contain elements very close to the screen margins and these parameter settings can already result in a huge distance to the original fixation location, e.g. at coordinate 1000 a linear factor of 0.9 or 1.1 already causes a difference of 100 px even without an additional offset. Hence, these ranges should cover all occurring errors, even if they are much larger than the maximum of 80 px found by Nüssli [2011, 54]. Errors that lie outside this already generous area should be handled with care and not treated automatically without inspection. If several parameters result in an equally optimal fitness score, the one that leaves the data closest to its original position is adopted, so the data is modified as little as possible.

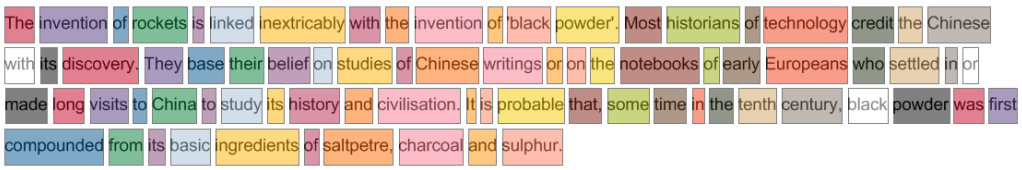
6.2.3.3 Adapting AOIs

Locations at which AOIs (see 7.2 *Areas of interest*) are not very dense will most likely have some fixations that land only close to an AOI. For example, fixations from reading the first line tend to be slightly above that line. Due to the size of the fovea the text is still located in the area of sharpest vision and since there are no other AOIs above, there is no need to force the gaze exactly onto the line in order to read it. The same occurs on very long lines that are surrounded by shorter ones and lines that only contain a single closing bracket. Such AOIs do not have to be looked at exactly, it is enough to look at their vicinity. Usually AOI borders comprise their respective item rather tightly, especially for words in a text. In order to take the increased fixation likelihood at less packed locations into account, the AOIs there are expanded, while the other AOIs retain their regular size.

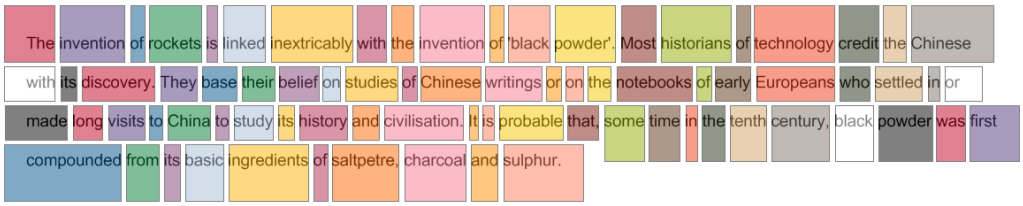
Extra margins are added to AOIs above the first and underneath the last line, and before the first and after the last element of every line. Within the text, extra margins are added where possible, e.g. in long lines surrounded by lots of empty space. The maximum amount added corresponds to circa 0.5° visual angle to ensure that the area covered by the expanded box is still well within foveal vision when targeting the respective AOI. The distance between AOI borders is kept comparable in size to the regular AOIs. Figures 6.2, 6.3, and 6.4 illustrate the differences between original and expanded AOIs. Inside the text the available empty space is evenly attributed among AOIs (figure 6.4). Due to the specific distribution of words and code elements in the EMCR stimuli, the expansion mostly affects the height of AOIs. This already helps to map more fixations to the correct AOI, since error tends to be greater vertically than horizontally [Hyrskykari, 2006, 667], [Nüssli, 2011, 64], [Špakov et al., 2018, 4-6]. Furthermore, SC stimuli offer more options for expansion, so their AOI structure is changed to a greater extent than those of NT stimuli.

6.2.3.4 Variants for evaluation

Both Nüssli and Lohmeier correct the whole trial at once. The EMCR study contains AOIs in the center and at the edges of the screen, so error will most likely vary locally. Therefore, it is tested whether correction can be improved by applying it individually to different screen regions. For this purpose, all fixations of a trial as well as local subsets are corrected. The variants *single* and *linear* technically both use the advanced error function 6.2 and apply it to the fixations of the whole trial. However, for *single*, the linear factor a is set to '1', so it also corresponds to error function 6.1.

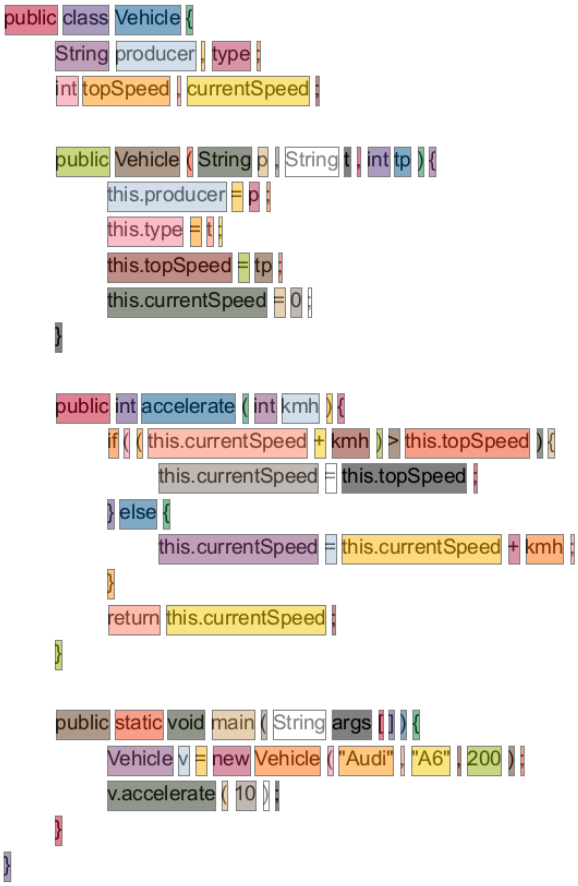


(a) Original

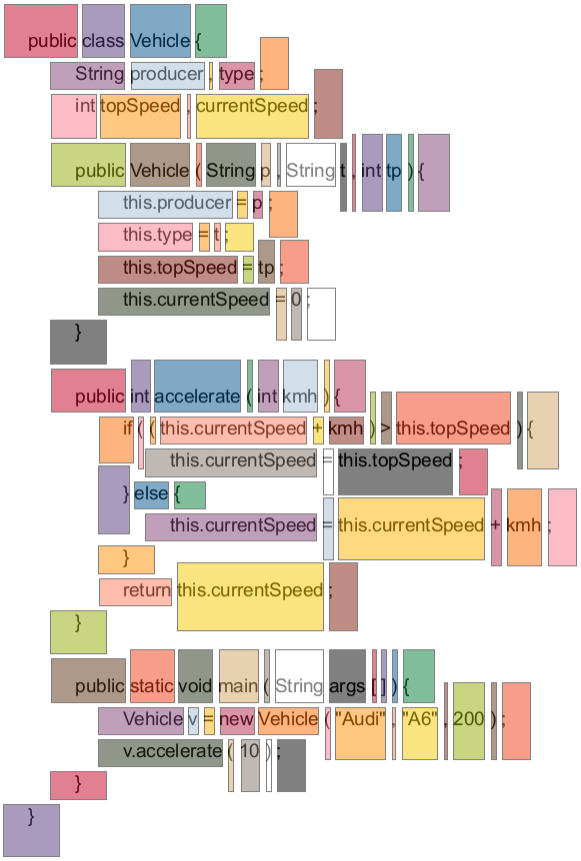


(b) Expanded

Figure 6.2: Exemplary NT AOIs

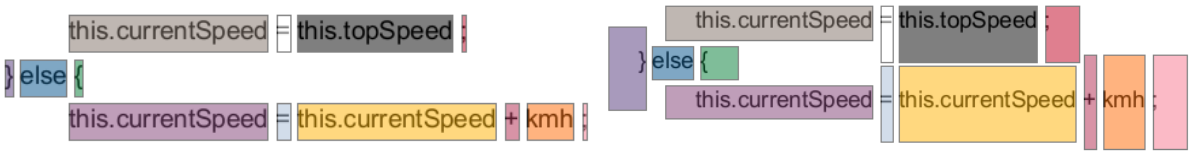


(a) Original



(b) Expanded

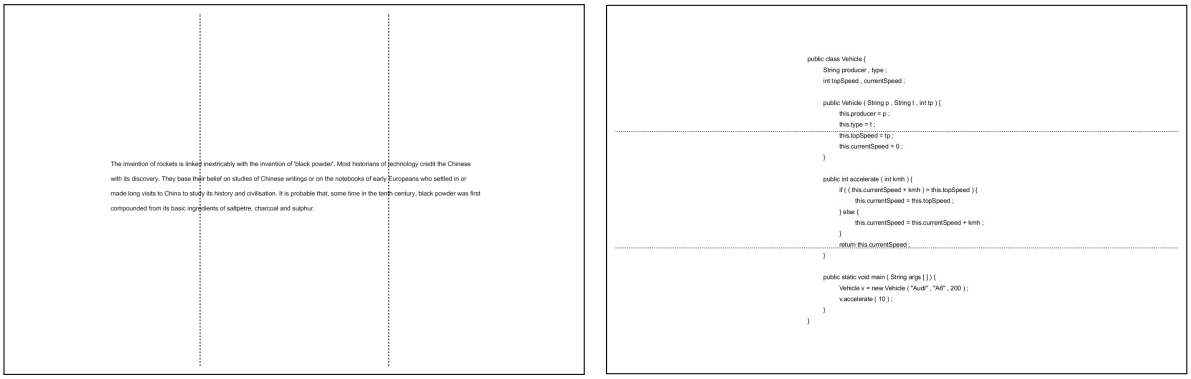
Figure 6.3: Exemplary SC AOIs



(a) Original (b) Expanded

Figure 6.4: Detailed illustration of dividing available space: With the original AOIs (a) method `accelerate` has a lot of empty space between the lines `this.currentSpeed = this.topSpeed` and `this.currentSpeed = this.currentSpeed + kmh`. In the expanded AOIs (b) half of the available space is added as margin underneath the upper line, half is added above the lower line.

Since error is usually rather small in the center and increases towards the edges, the screen will be split evenly into three sections: one in the middle and two at the edges. Two orientations of the split are tested: *horizontal* and *vertical*. Both are tested on all stimuli, but it is assumed that the most suitable orientation of the split depends on how the AOIs are spread. NT is located in the vertical middle of the screen, but stretches from left to right. Hence, it is expected that horizontal splitting works best for this type of text, as error is potentially larger at the sides, while the middle needs less correction (figure 6.5a). SC has only few long lines, but reaches close to the top and bottom of the screen and will probably benefit from vertical splits (figure 6.5b).



(a) NT with horizontal splits (b) SC with vertical splits

Figure 6.5: Splitting the screen into three subsections

If fixations are corrected independently for different screen regions, it might happen that fixations from neighboring regions are moved unreasonably far away from each other or heavily shifted into each other. This can especially occur when only few fixations are available for parameter optimization. To prevent this, a limitation is introduced (variants containing *limited*). First, optimal parameters are determined for the whole trial and serve as baseline. Then parameters are found for the three screen sections with the search space reduced to positions within circa $\pm$ 0.5° visual angle from that baseline. For variants including the linear factor, both the parameters from *single* and *linear* are tested as baseline (*limited_single* and *limited_linear*).

In addition, another approach will be tested, which weights the parameters from different regions (variants containing *weighted*). First, parameters are optimized for all horizontal and vertical sections individually and then weighted for every fixation according to the proximity between fixation location and screen region. However, NT often contains no fixations in the upper and lower sections, and SC mostly only fills the vertical dimension. So another variation is included (approaches containing *weighted_texttype*). For NT, parameters are computed for the whole trial and the three horizontally split sections, for SC, parameters are computed for

the whole trial and the three vertical sections. These four parameter sets are weighted. The weighting is also combined with the limitation.

In total, there are 22 variants of the correction approach to test, each with the original rather terse AOIs as well as with expanded AOIs. No time-based subsets of fixations are corrected. Gradual deterioration does not pose a serious threat, since most EMCR trials are rather short (median duration of ca. 1 min). In case error is building up in the few longer trials, the expanded AOIs already mitigate the impact of such changing offsets. Hence, generally splitting the trial into shorter segments at predefined points is hardly beneficial. Sudden changes in data quality that necessitate different parameters for correction than the gaze points until then are also an exception in the EMCR study. Consequently, error changing over time is not very relevant for the EMCR data. In contrast, error definitively varies over the screen, so correcting fixations based on screen region is much more promising. Combining region- and time-based splits will partly not leave enough fixations in a subset for reliable parameter optimization, so only region-based variants are pursued.

Variants with complete trial:

single An offset is computed for all fixations, i.e. $f_{corr} = f_{uncorr} + o$.

linear A linear factor and an offset are computed for all fixations, i.e. $f_{corr} = a * f_{uncorr} + o$.

Variants with subsets:

single_horizontal The screen is split into three horizontal sections, offsets are computed for each section separately.

single_horizontal_limited Same as single_horizontal, but only locations within bounds are tested.

single_vertical The screen is split into three vertical sections, offsets are computed for each section separately.

single_vertical_limited Same as single_vertical, but only locations within bounds are tested.

single_weighted The screen is split into three horizontal and three vertical sections. Offsets are calculated for each section separately and weighted according to the proximity of the uncorrected fixation location to each section.

single_weighted_limited Same as single_weighted, but only locations within bounds are tested.

single_weighted_texttype For NT the screen is split into three horizontal and one vertical section, for SC the screen is split into one horizontal and three vertical sections. Offsets are calculated for each section separately and weighted according to the proximity of the uncorrected fixation location to each section.

single_weighted_texttype_limited Same as single_weighted_texttype, but only locations within bounds are tested.

linear_horizontal The screen is split into three horizontal sections, a linear factor and an offset are computed for each section separately.

linear_horizontal_limited_single Same as linear_horizontal, but only locations within bounds of single are tested.

linear_horizontal_limited_linear Same as linear_horizontal, but only locations within bounds of linear are tested.

linear_vertical The screen is split into three vertical sections, a linear factor and an offset are computed for each section separately.

linear_vertical_limited_single Same as linear_vertical, but only locations within bounds of single are tested.

linear_vertical_limited_linear Same as linear_vertical, but only locations within bounds of linear are tested.

linear_weighted The screen is split into three horizontal and three vertical sections. A linear factor and an offset are computed for each section separately and weighted according to the proximity of the uncorrected fixation location to each section.

linear_weighted_limited_single Same as linear_weighted, but only locations within bounds of single are tested.

linear_weighted_limited_linear Same as linear_weighted, but only locations within bounds of linear are tested.

linear_weighted_texttype For NT the screen is split into three horizontal and one vertical section, for SC the screen is split into one horizontal and three vertical sections. A linear factor and an offset are computed for each section separately and weighted according to the proximity of the uncorrected fixation location to each section.

linear_weighted_texttype_limited_single Same as linear_weighted_texttype, but only locations within bounds of single are tested.

linear_weighted_texttype_limited_linear Same as linear_weighted_texttype, but only locations within bounds of linear are tested.

6.3 Evaluation using reference locations

Evaluating a correction method for eye tracking data poses a challenge. Using a more exact device like a scleral coil to obtain a ground truth is often not a feasible option. Therefore previous correction procedures were assessed with a variety of methods, e.g comparing the corrected data to data from manual correction or using different kinds of reference locations (see 5.4 *Evaluation approaches for error correction*).

RFLs provide very probable targets for certain fixations and are a good method to gauge errors and assess the performance of a correction method. Implicit RFLs arise when the target of a fixation can be deduced with high certainty, e.g. because the participant looks at a button and clicks it. For explicit RFLs, participants are specifically instructed to fixate a certain element. There are no implicit RFLs in the EMCR stimuli and explicit ones cannot be included, since they interfere with the comprehension process. However, the concept of RFLs can be adapted for evaluation.

An experiment was conducted especially for testing the different variants of the devised correction approach, in which participants were asked to look at a number of stimuli and use the mouse cursor to click at the location they are fixating. Then, fixation coordinates were compared to those provided by the clicks. These clicked locations are not part of the correction itself, they are used exclusively for evaluation purposes. To distinguish this solely evaluational usage of expected fixation locations from correctional RFLs, the coordinates of the clicks are called reference locations (RL). They provide a good approximation of the ground truth to choose the most suitable correction approaches, which will be evaluated further with data from the actual study and artificial data.

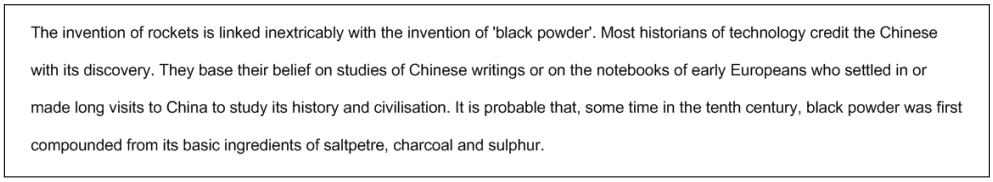
6.3.1 Stimuli

Two types of stimuli were used for evaluation. The texts are actual stimuli from the EMCR study. The grid stimuli present visual search tasks and were generated specifically for evaluation purposes. In total there are 12 stimuli, two with text and ten with grids.

6.3.1.1 Text

Two representative *texts* were chosen from the EMCR stimuli, one NT and one SC. The NTs used in the EMCR study are very similar in shape, position on the screen and number of lines. Since they are quite comparable, the first one was selected to simplify matters (figure 6.6a). In order to choose a good sample SC, the two programs were considered that were shown to both novices and experts. The longer of these programs was taken (novice L5_SC3 / expert SC3, see figure 6.6b), since it stretches closer to the edges and is therefore expected to bring on more complicated errors. Thereby it can be seen which errors will most likely occur in the study and made sure that the correction works even in difficult cases.

Participants in the evaluation study were instructed to read the text and to stop again and again to move the mouse onto a text element, fixate the cursor and simultaneously click. Two very different reading behaviors were tested to ensure that the correction does not depend on a certain viewing strategy. First the participants simulated linear reading by looking at the text from left to right, line after line. Then, participants "read" the text in random order, e.g. jumping back and forth between different parts of the text, and looking from right to left.



(a) NT

```
public class Vehicle {
        String producer , type ;
        int topSpeed , currentSpeed ;

        public Vehicle ( String p , String t , int tp ) {
                this.producer = p ;
                this.type = t ;
                this.topSpeed = tp ;
                this.currentSpeed = 0 ;
        }

        public int accelerate ( int kmh ) {
                if ( ( this.currentSpeed + kmh ) > this.topSpeed ) {
                        this.currentSpeed = this.topSpeed ;
                } else {
                        this.currentSpeed = this.currentSpeed + kmh ;
                }
                return this.currentSpeed ;
        }

        public static void main ( String args [ ] ) {
                Vehicle v = new Vehicle ( "Audi" , "A6" , 200 ) ;
                v.accelerate ( 10 ) ;
        }
}
```

(b) SC

Figure 6.6: Text stimuli

6.3.1.2 Grid

Additionally to the text stimuli, a second type of stimulus with a visual search task was employed, called *grid*. Participants had to find and click the ten letters from "a" to "j" in alphabetical order. NT and SC grids were constructed, whose structures are comparable to those of the texts (figure 6.7). In order to have the same shape as the corresponding text, targets were generated at the same coordinates as the first and last line of the text, as well as the leftmost and rightmost elements. The remaining target locations were drawn randomly from the respective text-AOIs. 64% of the NT-AOIs and 37% of the SC-AOIs were used in the grids. Hence, all grid-AOIs are located on the same screen positions as the text-AOIs and should cause comparable location-related errors. In order to obtain enough click data, five stimuli slides were generated per text type. These ten grids serve as additional test for the correction mechanism.

The underlying assumption of the correction approach is that gaze is mostly directed at AOIs, not blank space, hence it should also be applicable to other visual tasks than reading. Since the screen locations of the grid targets are the same as for the texts, it can be evaluated, if the tested approaches are able to correct errors that change due to the screen locations of the AOIs. Furthermore, grid-AOIs are much less dense, so identifying the fixated element is more straightforward as in text stimuli. Moreover, participants reading and clicking the text stimuli reported finding it quite challenging to keep looking at the word they are trying to click. When inspecting the gaze replay on text stimuli, instances were found, in which gaze and cursor were directed at a word, but right before clicking, the eyes moved on (see 6.3.3.1 *Reference locations*). Practiced readers are accustomed to keep moving forward during reading and participants had to suppress these usual forward eye movements in order to click the text element. The search tasks in the grid stimuli require a different visual behavior than reading and have less elements, so it is easier for participants to stay on a target while clicking, thus providing very reliable reference locations.

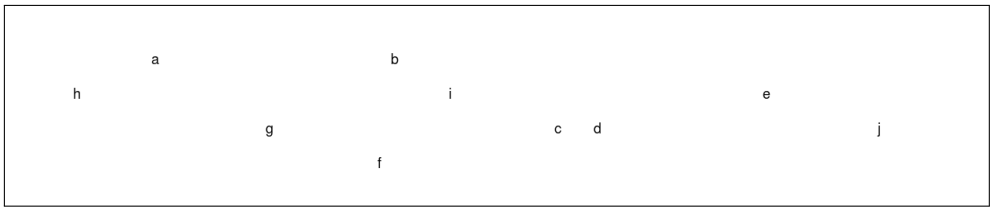
6.3.2 Recording situation and participants

To ensure that the error is not specific to a certain device, two different eye trackers were tested, the SMI RED-m 120Hz tracker that is also used in the EMCR study and the SMI RED250mobile 250Hz tracker. The experiment was set up and recorded with SMI Experiment Center version 3.7.42, iViewX version 4.2.1.0. The recording situation was as close as possible to the one in the actual study. The main sources of error in the study are free head movements and changing light conditions. Thus these conditions were kept for the evaluation experiment.

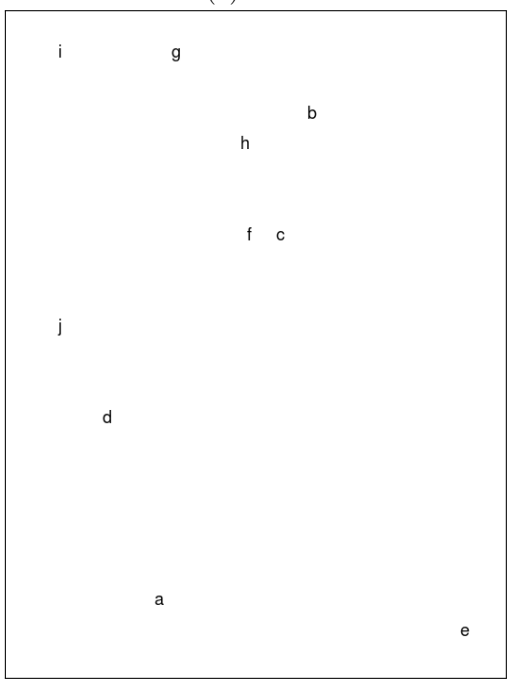
Three participants were recruited, but one had to be excluded due to low data quality. Of the two remaining participants, one had normal vision, the other wore glasses. One person was blue-eyed, the other brown-eyed. Even though these are very few participants, they differed in several key eye features. Furthermore, this experiment is only the first part of the evaluation. The correction will be tested further with a large part of the EMCR data as well as artificially created scanpaths.

6.3.3 Analysis procedure and results

The trials recorded in the evaluation experiment last between 18 and 108 sec, with a median duration of 36 sec [26..48]. On average, trials with text as stimulus were longer (median=66 sec [46..72]) than those with grid stimuli (median=32 sec [23..41]). The longer duration is most likely caused by the much greater number of AOIs in the text stimuli. For analysis, it is first necessary to identify which fixations and RLs belong together. Based on these pairs, the occurring errors are measured and the performance of the different correction variants is assessed.



(a) NT



(b) SC

Figure 6.7: Exemplary grid stimuli

Errors are calculated as Euclidean distance between the two screen coordinates. Following Nüssli's terminology, *fitness* is used to denote the percentage of fixations that land on an AOI, regardless of whether it is the correct one or not. The percentage of fixations that are on the same AOI as the mouse click is referred to as *match*. Several categories of data will be analyzed in detail:

- Stimulus type: text, grid
- Text type: nt, sc
- Device: 120Hz, 250Hz

6.3.3.1 Reference locations

The intuitive way to establish which fixation belongs to a certain reference location would be to simply use the fixation that occurs at the same time as the click. However, there are several difficulties with this approach. Firstly, for some of the clicks no simultaneous fixation was recorded. This can be due to the participant actually not fixating while clicking or because of a recording malfunction, e.g. momentary signal loss. However, these few clicks can just be omitted. Much more problematic are cases in which a fixation was recorded simultaneously with the mouse click, but at a location unreasonably far away from it. Replays of the raw data revealed a number of instances, where the gaze hovered over the mouse cursor, but jerked away right before the actual click occurred. This is in line with participants stating that they partly

found it challenging to focus on the cursor while clicking. A probable explanation is that after fixating the mouse cursor the eyes already move on, even though the clicking action was not yet executed. Hornof & Halverson [2002, 602] also mention the possibility of such situations. Especially in reading the eyes are accustomed to continually moving forward, so the constant stopping is an unusual task. Therefore, in addition to the fixation that occurs together with the click, the one right before it is also inspected. Clicks are only accepted for evaluation, if they have a preceding fixation within three seconds (pre) as well as a simultaneous (sim) one. The time interval for pre-fixations was restricted, since it is very unlikely than fixations that occurred earlier than that can still belong to the click. Of these two fixations the one with the shortest spatial distance to the click location is taken as belonging to the RL and thus used for evaluation (eval). The reasoning behind this approach is that if the person actually looks at the mouse pointer while clicking, the simultaneous fixation is closer to the click location than the preceding fixation. In case both preceding and simultaneous fixation cluster around the cursor, because the participant looked at the cursor for some time, these two fixations are largely interchangeable. Since both fixations are very close in time and space, comparable errors can be expected. If however the eyes already moved on, the simultaneous fixation will be farther away and the preceding one is used for evaluation instead.

In total, 696 clicks were recorded, 296 on text stimuli and 400 on grids. 656 (94%) of these have both a valid preceding and simultaneous fixation and can thus be used for evaluation (292 for texts and 364 for grids). Simultaneous fixations are predominantly closer to the click location than the preceding ones. So overall participants were able to keep their gaze close to the cursor while clicking (see table 6.1).

	total	valid_pre	valid_sim	valid_eval	eval_pre	eval_sim
all	696	659 (95%)	691 (99%)	656 (94%)	177 (27%)	479 (73%)
text	296	294 (99%)	294 (99%)	292 (99%)	72 (25%)	220 (75%)
grid	400	365 (91%)	397 (99%)	364 (91%)	105 (29%)	259 (71%)
nt	309	294 (95%)	307 (99%)	292 (94%)	75 (26%)	217 (74%)
sc	387	365 (94%)	384 (99%)	364 (94%)	102 (28%)	262 (72%)
text_nt	109	109 (100%)	108 (99%)	108 (99%)	19 (18%)	89 (82%)
text_sc	187	185 (99%)	186 (99%)	184 (98%)	53 (29%)	131 (71%)
grid_nt	200	185 (92%)	199 (100%)	184 (92%)	56 (30%)	128 (70%)
grid_sc	200	180 (90%)	198 (99%)	180 (90%)	49 (27%)	131 (73%)
120	343	324 (94%)	338 (99%)	321 (94%)	64 (20%)	257 (80%)
250	353	335 (95%)	353 (100%)	335 (95%)	113 (34%)	222 (66%)
text_nt_120	58	58 (100%)	57 (98%)	57 (98%)	9 (16%)	48 (84%)
text_sc_120	85	85 (100%)	84 (99%)	84 (99%)	23 (27%)	61 (73%)

Table 6.1: Number and percentages of reference locations and associated fixations for categories of most interest: Pre-fixations occur right before the click, sim-fixations simultaneously. For locations with both a valid pre- and sim-fixation, the one closest to the click location is used as eval-fixation. The last two columns show the share of pre- and sim-fixations in eval-fixations.

Also, the summary of distances between RLs and fixations (table 6.2), as well as figure 6.8 show that preceding fixations are generally much farther away from the click than simultaneous fixations (median distance pre-fixations=29 px [17..55], median distance sim-fixations=17 px [11..25]). Even though simultaneous fixations are generally located close to the clicks, some of their distances are extreme outliers (greater than the 3 IQR) and the maximum offset is unreasonably high (170 px). In the eval-condition, where pre- and sim-fixations are combined,

there are no extreme outliers anymore and the maximum distance is reduced to 55 px. Hence, while the simultaneous fixations are a better choice than the preceding ones, their combination results in the most plausible RLs. All RLs with a valid eval-fixation were excepted. No manual selection of RLs was employed to avoid bias. Besides, due to the over 600 RLs even a few cases of RL and fixation not being a perfect pairing are of almost no consequence.

Fixation type	Minimum	1st Quartile	Median	Mean	3rd Quartile	Maximum
pre	1	17	29	52	55	608
sim	1	11	17	19	25	170
eval	1	9	15	16	22	55

Table 6.2: Summary of distances between reference and fixation location (in pixels) for the fixation types pre, sim, and eval

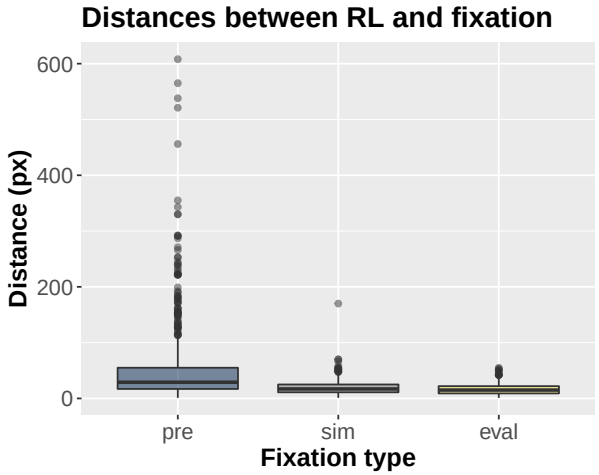


Figure 6.8: Distances between reference and fixation locations for the fixation types pre, sim and eval

6.3.3.2 Errors

The distances between click and fixation locations provide the errors for evaluation. Figure 6.9 shows the distribution of error values, table 6.3 summarizes the errors found in the most relevant categories. The median overall error is 15 px [9..22], which corresponds to ca. 0.4° of visual angle at a typical viewing distance of 65 cm. This in line with what is to be expected for the two eye trackers that were used. The accuracy of the 120Hz device is listed as 0.5° [SensoMotoric Instruments, 2016a] and for the 250Hz device it is 0.4° [SensoMotoric Instruments, 2016b]. The maximum error is 55 px, so the chosen parameter ranges are more than sufficiently large to correct these errors.

The error values are not normally distributed, thus Mann-Whitney tests were applied to compare data from the categories stimulus type, text type and device. The p-values were adjusted to multiple testing, however this did not change the outcome. Fixations on text stimuli exhibit significantly smaller errors than those on grid stimuli ($p<0.001$). This might be caused by the plenty of empty space around the grid AOIs. The letters in the search task constitute only small AOIs and there are hardly any other targets close-by. Thus they can be perceived very well by merely looking near them without centering the gaze exactly onto them. Neither the

differences between NT and SC stimuli, nor between the two eye trackers are significant. Hence the data from both devices is comparable. However, there is a significant difference between the errors found in the NT- and SC-text stimuli, while there is no such difference between the NT- and SC-grids. This finding indicates that the optimal correction method might differ for these two text types. In addition, no difference was found between errors from linear and random reading in the text stimuli (table 6.4).

Category	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
all	1	9	15	16	22	55
text	1	8	13	14	20	52
grid	1	10	17	18	23	55
nt	2	9	14	17	23	55
sc	1	8	15	16	22	52
text_nt	2	10	14	16	20	42
text_sc	1	6	11	14	18	52
grid_nt	2	9	15	18	24	55
grid_sc	1	12	17	17	23	41
120	1	8	14	16	22	52
250	1	9	16	17	23	55
text_nt_120	2	10	15	17	25	42
text_sc_120	1	6	10	13	18	52

Table 6.3: Errors for categories of most interest (in pixels)

Comparison	p
text - grid	**<0.001**
nt - sc	0.232
text_nt - text_sc	**0.028**
grid_nt - grid_sc	0.414
120 - 250	0.202
linear - random	0.254
horizontal - vertical	**<0.001**
abs(horizontal) - abs(vertical)	**<0.001**

Table 6.4: Differences between errors in categories of most interest

Next, horizontal and vertical errors are compared (see figures 6.10 and 6.11). Neither the directional nor the absolute horizontal and vertical error values are normally distributed, so a Wilcoxon matched-pairs test was applied. Both directional and absolute horizontal and vertical errors are significantly different from each other, with horizontal error being smaller than vertical (see table 6.4 and figure 6.11). This is consistent with the findings by Feit et al. [2017, 1121,1122], Hyrskykari [2006, 667], and Nüssli [2011, 64], who also state that error tends to be greater vertically than horizontally. However, since both horizontal and vertical error will be corrected automatically with an ample parameter range, at least for correction and evaluation it is hardly of any consequence which error is greater. It is however relevant that the error does indeed change over the screen. Figure 6.12 shows how much both horizontal and vertical errors vary depending on screen-coordinate. This demonstrates quite clearly that neither an offset-only approach nor the combination of linear factor and offset are sufficient to address these non-linear errors.

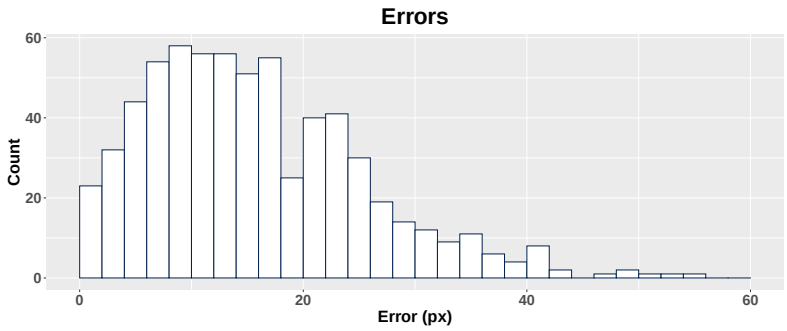


Figure 6.9: Overall absolute errors based on RLs

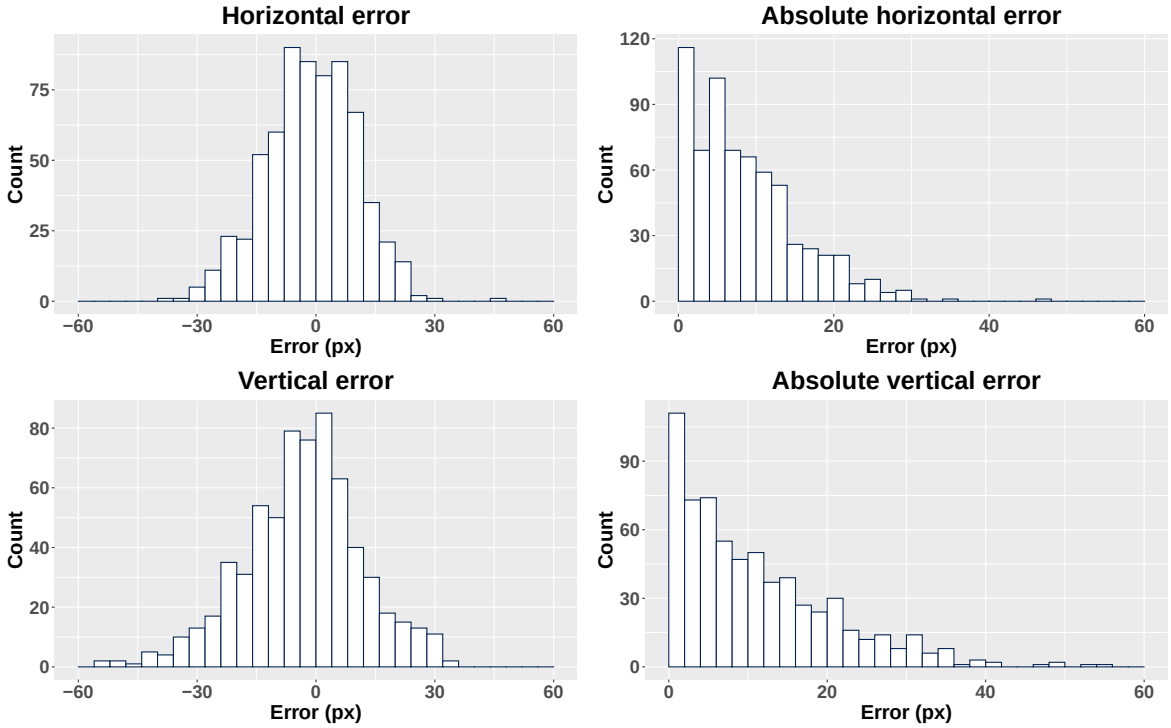
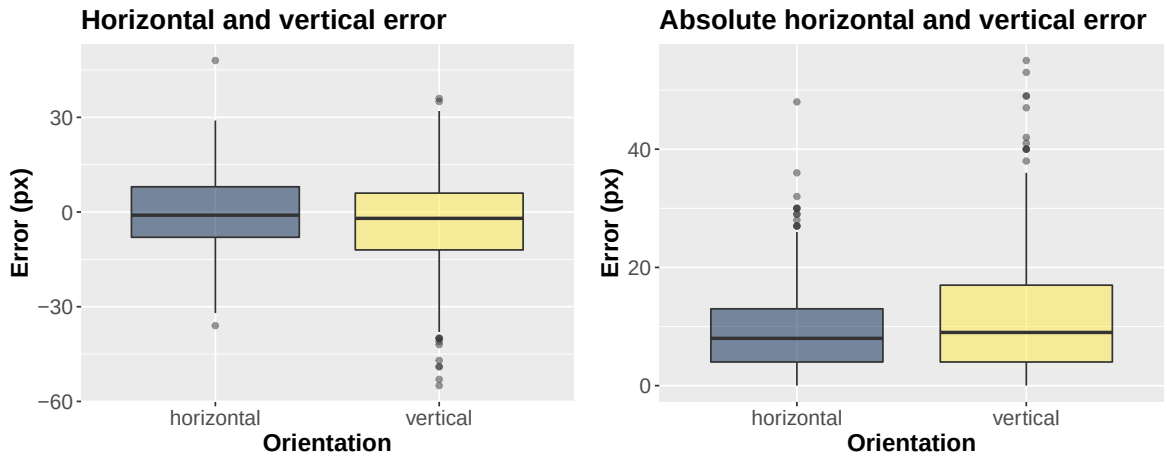


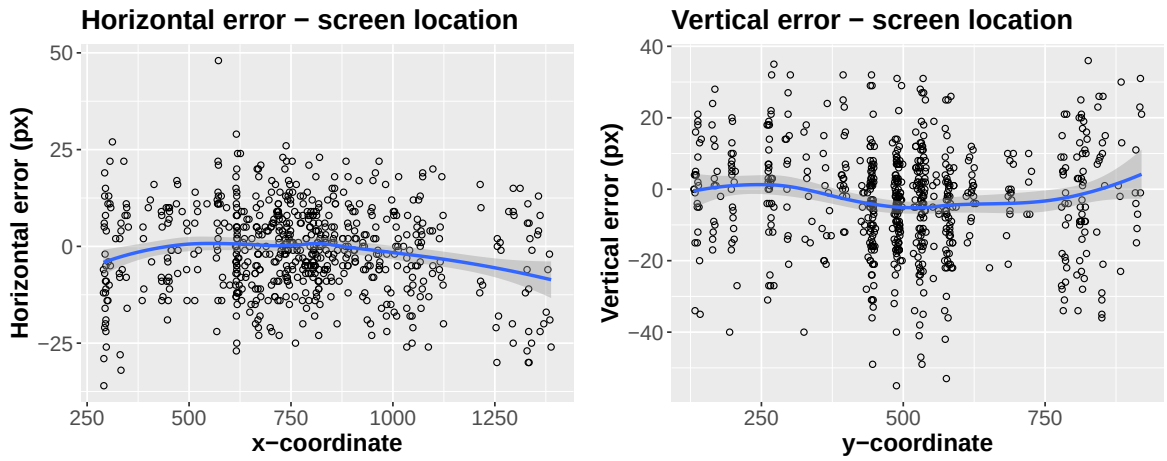
Figure 6.10: Distribution of horizontal and vertical errors

Additionally to the error values, it is of interest what percentage of the fixations was recorded on the same AOI as the click (figure 6.13 and table 6.5). Without correction, only 53% of the fixations were on the correct line, 40% on the correct element. Since grid stimuli include a greater error than the texts, it is not surprising that the percentage of correct matches is lower for grids (37% on lines and 18% on elements) than for texts (72% on lines and 69% on elements). The reason for the fewer correct matches in grid-stimuli is most likely the small AOI sizes. The texts contain a number of longer words and thus bigger AOIs than the one-letter grid-targets. Thus, for grids even a small error will result in the fixation being recorded on the wrong or no element, while for the texts there is a good chance that the fixation still hits the correct AOI. Furthermore, there are more correct matches for NT (60% on lines and 43% on elements) than for SC (46% on lines and 38% on elements). Again, this is probably due to AOI size. The SC text has shorter lines than NT text and in addition, it includes many short elements with only one or two letters, e.g. separators and operators. Moreover, the NT-grids partly also have longer line-AOIs than the SC-grids.



(a) Directional errors (b) Absolute errors

Figure 6.11: Horizontal and vertical errors



(a) Horizontal (b) Vertical

Figure 6.12: Error relating to screen location

Figure 6.13 shows that expanding the AOIs as described in 6.2.3.3 *Adapting AOIs* alone increased the percentage of correct matches immensely, on line-level it rises from 53% to 90%, on element-level from 40% to 87%. Expanding the AOIs had the biggest impact on grid-data (37% to 93% on lines and 18% to 91% on elements), but also considerably increased the correctness for text-data (72% to 86% on lines and 69% to 81% on elements). The impact on text-data is of most interest, since it is the most representative for the stimuli used in the actual study, while the grid was included for evaluation purposes only. Comparing text-NT and text-SC, the expansion is good for both, but the SC profits most, with the percentage of correct matches increasing from 64% to 82% on lines and from 60% to 76% on elements. For text-NT there is also an improvement, but smaller than for the other stimuli: 86% to 93% on lines and 84% to 91% on elements. This supports the notion that the small AOIs largely contribute to the low matches for grid and text-SC stimuli with original AOIs (see table 6.5).

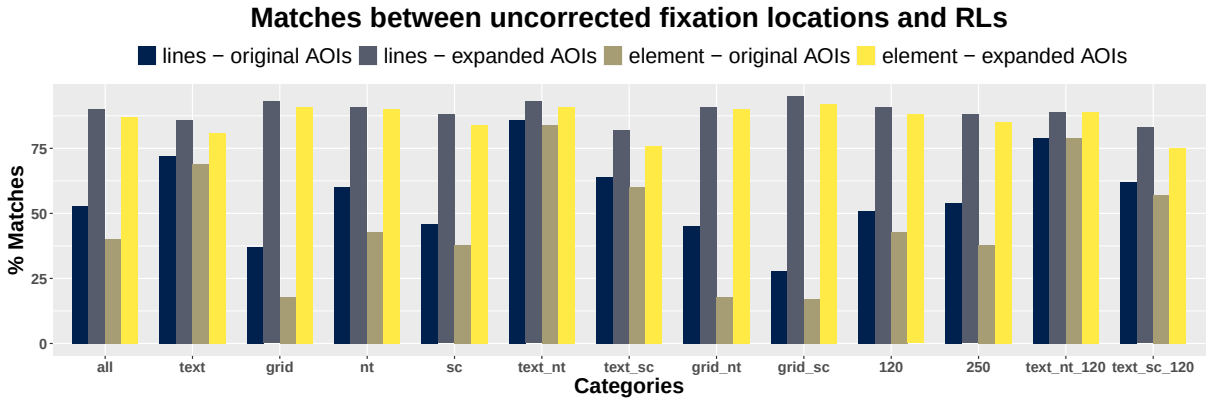


Figure 6.13: Percentage of uncorrected fixations on the same AOI as the click, when using original and expanded AOIs

6.3.3.3 Correction

The fixation data from the evaluation experiment was corrected using all variants described in section 6.2.3.4 *Variants for evaluation*. In order to determine the fitness of each screen location, a matrix of the same size as the stimulus screen was created, which contains '1' for every pixel which belongs to an AOI and '0' otherwise. Since there are two types of AOIs, correction was performed twice, once with original and once with expanded AOIs. The AOI type is also relevant when testing whether a fixation is on the same AOI as the corresponding click, so both types are employed for evaluation as well. Altogether three different AOI combinations were used for obtaining the fitness values during correction and performance evaluation:

- Original AOIs for correction and evaluation
- Original AOIs for correction, but expanded AOIs for evaluation (*expeval*)
- Expanded AOIs for correction and evaluation (*exp*)

Due to the huge computational effort of testing all correction variants with both AOI types, the calculations were distributed over a computing cluster.[1]

After correction, the new fixation positions were mapped to AOIs and compared to the AOIs that were clicked. This serves as main measure for evaluation. Furthermore, the distances between corrected fixation locations and their respective RL are analyzed. However, the fixation location provides only the center of the fixation, so additionally to using just a single pixel on the screen, the area around this point is inspected. The fovea covers an area of about 1.5 - 2° visual angle, in which the fixated target is perceived with very high visual acuity [Holmqvist et al., 2011, 21], [Rayner, 1998, 374], [Rayner et al., 2005, 85]. Hence not only the element at the exact fixation position is seen, but also the surrounding ones within the foveal area. Additionally, the fixation coordinates specify only the center of the raw data points that were merged into a fixation. In order to gauge whether the recorded fixation location is so close to the correct target that this item is definitively perceived, an area of roughly 0.5° around the center of the fixation is examined. The smaller angle of 0.5° instead of 2° is chosen to have an area in which it is safe to assume that the gaze was hovering during the fixation. It also corresponds to about half of the average dispersion in x- and y-direction found in the fixations used for evaluation, so the raw data definitively spreads within this area. Table 6.5 gives detailed results for the most interesting subsets of variants and also lists the percentage of cases, in which the correct

[1]The author gratefully acknowledges the funding of this project by computing time provided by the Paderborn Center for Parallel Computing and the Department of Mathematics and Computer Science at Freie Universität Berlin.

element is within this *scope* (see figure 6.14). This measure is especially interesting for stimuli with many small AOIs, for which even very small errors can lead to the fixation being assigned to the wrong AOI. When analyzing the EMCR data, the same approach will be used to assess which elements were looked at.

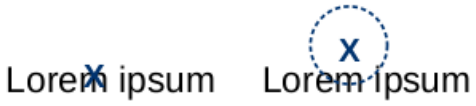


Figure 6.14: Illustration of *match* (left) and *scope* (right).

Finally, fitness values before and after correction are provided, i.e. the percentage of fixations on an AOI. In some cases, the number of correct matches drops, even though the fitness value increases, e.g. for text stimuli the percentage of correct matches on elements decreases from 69% for uncorrected data to 57% when using the *linear* approach, even though the fitness increases from 71% to 85%. Hence fitness alone is not sufficient to evaluate the performance. Therefore a combination of several measures is used.

The approaches that perform best overall and separately per stimulus type (text and grid), text type (NT and SC), and finally for the categories text_nt, text_sc, grid_nt, grid_sc, text_nt_120, and text_sc_120 were scrutinized in detail to find the optimal methods. Since no difference was found in the errors from linear and random text reading, and from the two eye tracking devices, their individual correction results are not discussed further.

The variants *single* and *linear*, which correct the complete trial at once, result in only slightly improved data. For the *single* approach the percentage of correct matches rises from 53% to 67% on line-level and from 40% to 55% for elements. When applying the *linear* approach, 75% of the corrected fixations are on the correct line and 58% on the correct element. However, only expanding the AOIs without any correction already results in 90% correct matches on lines and 87% on elements. Hence, even though both approaches increase the amount of fixations on the correct AOI, their impact is rather small. Combining correction and expanded AOIs (expeval) provides much better results. For the *single* approach 91% of the corrected fixations are on the correct line, 85% on the correct element. For the *linear* variant, 92% of the fixations match on lines, 81% on elements. However, the good results are mostly due to the grid-stimuli. On text, even with expanded AOIs, *single* reaches only 85% matches on lines and 72% on elements, *linear* 87% on lines and 63% on elements.

Approaches that split the data across the screen are a better choice. To find the most suitable option, only approaches with a limited range were taken into final consideration. Even though the results of the limited and unlimited version of an approach are often identical, there is a significant difference between their performance on line-level ($p=0.036$) with limited being more correct. Furthermore, using a limited approach ensures that only valid parameters are computed. While the limitation did not make a huge difference for the evaluation data, it is very likely that when correction is applied to a larger data set, much more cases will occur in which fixations from one screen section would be moved unreasonably in relation to the other sections, especially, if only few data points are available for optimization.

6.3.4 Chosen approaches

NT

For NT the approach single_horizontal_limited performed best and will be called *correction_nt*. It uses a linear factor of '1', so there is no scaling. First an offset is calculated for the whole trial. Then the stimulus is split into three horizontal sections (figure 6.5a) and an offset is determined

for fixations in each section individually. The parameter range for this second step is reduced to the bounds provided by the overall offset. Using the original terse AOIs during correction yields slightly better results than expanded AOIs, so the latter are not detailed. However, for evaluation the AOI at the corrected fixation location is obtained by expanded AOIs (expeval). Scaling with a linear factor had no additional value for NT, even though the stimulus elements stretch to the left and right screen margins. This is probably because they are located in the vertical middle of the screen and horizontal error was found to be significantly less than vertical error, so NT elements are not as much affected by error. Besides, the actual NT-texts contain a good number of longer elements, so the gaze often lands on the correct AOI despite a small offset.

The results for the category text_nt are of most interest. Inspecting correct matches on line-level shows that for uncorrected data 86% of the fixations are on the correct line. When expanding the AOIs for evaluation, the percentage of correct matches rises to 93%. However, after correction the portion of fixations on the correct line increases further to 99%. Taking the area around the fixation into account, 97% of the uncorrected and 100% of the corrected fixations have the correct line within the area of highest visual acuity. On word-level 84% of the uncorrected fixations are on the correct element, with expanded AOIs 91%. After correction the matches rise to 95%. For 97% of uncorrected and 99% of corrected fixations, the correct word is within scope. Before correction the median error for this category is 14 px [10..20]. After correction it slightly drops to 12 px [8..18].

Correction_nt performs very well not only for text_nt, but also in all other NT-related categories. When taking only the subset of text_nt that was recorded with the 120Hz eye tracker which was also used in the EMCR study, 98% of the corrected fixations are on the correct line, 95% on the correct word. Looking at the area around the fixation, 100% of the fixations have both the correct line and word within scope. For all nt-stimuli, which consist of text_nt and grid_nt, 98% of the corrected fixations are on the correct line, 96% on the correct element. Similarly, in grid_nt, 97% of the corrected fixations are on the correct line and element (table 6.5). Consequently, correction results in highly accurate NT data.

SC

For SC the approach linear_vertical_limited_linear with expanded AOIs during correction as well as evaluation (exp) performed best and will be called *correction_sc*. First a linear factor and an offset are calculated for the whole trial. Then a linear factor and an offset are computed for the fixations in each of the three vertical sections (fig 6.5b) using the reduced range determined by the overall parameters.

For this method, the results for the category text_sc are of most interest. 64% of the uncorrected fixations are on the correct line, 82% when expanded AOIs are used for comparison. After correction, the number of line-matches rises to 89%. The portion of fixations that have the correct line in scope increases from 88% to 92% due to correction. For elements, 60% of the uncorrected fixations match on the correct target, 76% when using expanded AOIs. After correction 78% of the fixations fall on the correct element. The percentage of fixations that have the correct element within scope also increases from 87% to 90%. The median error in the category text_sc actually increases due to correction, from 11 px [6..18] to 12 px [7..18], even though the data gets more correct. The increased distance is caused by variable errors and the linear factor. In order to get more fixations onto the correct AOI, some fixations are shifted slightly away from the click location, but only within the margins of the correct AOI.

Inspecting the results for text_sc with the 120Hz eye tracker shows an increased match rate of 89% on lines and 79% on elements, 90% have the correct line in scope, 85% the correct element. For the SC-stimuli, consisting of text_sc and grid_sc, after correction 93% of the fixations are on the correct line, 87% on the correct element. Finally, for grid_sc the fixations

match on 97% of the lines and 96% of the elements (table 6.5).

6.3.5 Conclusion

All variations of the error correction method outlined in 6.2.3.4 *Variants for evaluation* were evaluated using reference locations. Two approaches were identified as suitable for the EMCR data, one for NT and one for SC. A combination of several representative types of stimuli, texts and tasks were employed to ensure that the correction is applicable to the EMCR data.

Overall evaluation results show that due to correction and using the adapted AOIs, the percentage of fixations on the correct line increases from 53% to 95% and on the correct element from 40% to 91%. This constitutes an immense improvement. When considering only text-stimuli, the portion of correct matches rises from 72% to 92% on lines and from 69% to 85% on elements. Differentiating between text types, for NT-texts correctness on line-level reaches 99%, on word-level 95%. For SC-text, correctness rises to 89% on lines and 78% on elements. Taking the scope around fixations into account, the resulting matches are even higher, 100% on lines and 99% on elements for NT-texts, and 92% on lines and 90% on elements for SC-texts. Correction is a bit less successful for SC than for NT, but still considerably improves the correctness of the fixation locations. In the analyses of the EMCR data, correctness is mostly relevant on line-level, so these results are entirely sufficient.

Analyzing RL data by many more participants might have yielded another possible variant of the correction method. However, the chosen methods correction_nt and correction_sc are definitively appropriate for their respective task and will be assessed further using data from the EMCR study and artificial data.

6.4 Evaluation using manually corrected data

A large subset of the data collected in the EMCR study was corrected manually and can be compared to the results of the chosen automatic correction methods. In order to make manual correction as objective as possible, it was carried out by a group of researchers. Each trial was handled by two people collaboratively, whereat one person was the same for all trials [Busjahn et al., 2015a, 260].

6.4.1 Data

The manually corrected dataset contains fixations from 18 programmers (12 novices and 6 experts). It consists of 118 trials, 15 NT and 103 SC (82 from novices, 21 from experts) and incorporates 27 stimuli (3 NT, 24 SC). The stimuli cover all NTs and SCs from the EMCR study. In addition to the stimuli that will be used for analysis, this evaluation dataset contains several extra SCs, e.g. the ones recorded to document the students' progress over the Java course (see 3.2.2.1 *Novice programmers*) and are included here to have an even broader data basis for evaluation. The trials last between 6 and 271 sec and have an average duration of almost a minute (median=50 sec [24..86]).

The data contains 24,019 fixations, of which 23,932 are valid, i.e. within screen dimensions. During manual correction, 68% of the valid fixations were changed. However, the modifications are not equally distributed among text types. Only 29% of NT fixations were edited, but 74% for SC. In contrast, automatic correction, using correction_nt for NT and correction_sc for SC, affected almost all fixations. For the two expertise groups the portion of changed fixations is comparable, 75% of the novice and 71% of the expert data was changed during manual correction and almost all during automatic correction. After manual correction 15% of the fixations do not land on an element and therefore do not provide an AOI for comparison. These fixations have only a minor value for evaluation and to a certain degree dilute the results. Hence, only fixations are considered for evaluation which are valid and have an AOI at the manually corrected location (valid_aoi). This still leaves 20,340 fixations for comparison, 2,982 for NT and 17,358 for SC (12,753 from novices, 4,605 from experts), see table 6.6.

6.4.2 Analysis procedure and results

Manually corrected fixations serve as basis for comparison during evaluation. The term *match* denotes the percentage of fixations on the same AOI as the manually corrected ones, *scope* indicates how many

	median	line			element		
	distance	match	scope	fitness	match	scope	fitness
all_uncorr	15	53	84	50	40	81	36
all_uncorr_expeval	15	90	84	70	87	81	79
all_single	14	67	90	62	55	87	49
all_single_expeval	14	91	90	76	85	87	83
all_linear	13	75	92	68	58	84	55
all_linear_expeval	13	92	92	79	81	84	85
text_uncorr	13	72	91	75	69	91	71
text_uncorr_expeval	13	86	91	84	81	91	88
text_single	16	71	93	80	60	88	78
text_single_expeval	16	85	93	86	72	88	90
text_linear	17	80	94	86	57	80	85
text_linear_expeval	17	87	94	89	63	80	93
grid_uncorr	17	37	79	32	18	73	9
grid_uncorr_expeval	17	93	79	60	91	73	73
grid_single	12	65	88	50	52	85	28
grid_single_expeval	12	96	88	69	95	85	78
grid_linear	10	70	90	54	58	87	33
grid_linear_expeval	10	96	90	72	96	87	79
nt_uncorr	14	60	88	63	43	84	38
nt_uncorr_expeval	14	91	88	81	90	84	81
nt_correction_nt	11	98	96	89	96	94	88
sc_uncorr	15	46	81	39	38	79	33
sc_uncorr_expeval	15	88	81	61	84	79	78
sc_correction_sc	14	93	86	68	87	83	92
text_nt_uncorr	14	86	97	87	84	97	83
text_nt_uncorr_expeval	14	93	97	92	91	97	92
text_nt_single_expeval	16	94	94	92	89	93	94
text_nt_linear_expeval	19	95	100	94	69	80	98
text_nt_correction_nt	12	99	100	95	95	99	99
text_sc_uncorr	11	64	88	66	60	87	63
text_sc_uncorr_expeval	11	82	88	79	76	87	85
text_sc_single_expeval	15	80	92	81	62	86	87
text_sc_linear_expeval	16	83	91	85	60	80	89
text_sc_correction_sc	12	89	92	89	78	90	99
grid_nt_uncorr	15	45	83	48	18	76	10
grid_nt_uncorr_expeval	15	91	83	75	90	76	74
grid_nt_correction_nt	10	97	94	86	97	91	81
grid_sc_uncorr	17	28	74	16	17	70	9
grid_sc_uncorr_expeval	17	95	74	45	92	70	72
grid_sc_correction_sc	17	97	79	50	96	76	87
text_nt_120_uncorr	15	79	95	87	79	95	82
text_nt_120_uncorr_expeval	15	89	95	92	89	95	92
text_nt_120_correction_nt	12	98	100	95	95	100	98
text_sc_120_uncorr	10	62	87	65	57	85	63
text_sc_120_uncorr_expeval	10	83	87	80	75	85	86
text_sc_120_correction_sc	11	89	90	87	79	85	99

Table 6.5: Detailed correction results for the most relevant categories and subsets: median distance (in pixels), percentages of fixations on the correct AOI (match), within scope of the correct AOI, and fitness in %, each on line- and element-level. Results for the two chosen variants are highlighted.

Fixations	All	NT	SC	Novices	Experts
total number	24019	3394	20625	14833	5792
valid	23932	3394	20538	14826	5712
valid_aoi	20340	2982	17358	12753	4605
manually modified	16192	998	15194	11119	4075
automatically modified	23816	3394	20422	14714	5708

Table 6.6: Subset of EMCR fixations that was corrected both manually and automatically: Valid fixations are within screen dimensions, valid_aoi fixations are within screen dimensions and have an element at the manually corrected fixation location

percent of the fixations have this AOI within ca. 0.5° visual angle (see figure 6.14). In the previous evaluation step expanded AOIs proved to be more useful than the terse original ones, so only results for expanded AOIs are included (table 6.7). To facilitate reading, the following identifiers are used for the different fixation sets:

- uncorrected fixations: fixations_uncorr
- manually corrected fixations: fixations_man
- automatically corrected fixations: fixations_auto

The distances between fixations_uncorr and fixations_man, as well as between fixations_auto and fixations_man are not called errors, since manually corrected fixations only provide a rough estimation of the correct locations. Again, Euclidean distances between the fixations' screen locations are used as distance metric. The median distance between fixations_uncorr and fixations_man is 10 px [0..27], between fixations_uncorr and fixations_auto 22 px [12..43]. Thus, automatic correction moves the fixations significantly farther away from the original position than manual correction does ($p<0.001$). For comparison, the median distance between the corrected datasets fixations_man and fixations_auto is 18 px [9..32]. Overall, automatic correction brought more fixations to the same line as fixations_man (70% uncorrected to 84% automatically corrected), but slightly less fixations match on the element (68% uncorrected to 65% automatically corrected). However, the portion of fixations that have the same AOI within scope increases both on lines (75% to 88%) and on elements (73% to 75%), see table 6.7.

For NT, fixations_uncorr and fixations_man agree quite well. Since most NT fixations in fixations_uncorr and fixations_man are identical, their median distance is 0 px [0..5], between fixations_auto and fixations_man it is 13 px [6..21]. Also, both on line- and element-level, the percentage of matches is higher for uncorrected data (95% on both AOI-levels) than for the automatically corrected data (93% for lines and 77% for elements). The same is found for the scope, where rates are slightly higher for uncorrected data (97% on both AOI-levels), than for automatically corrected data (96% on lines, 89% on elements).

The uncorrected SC fixations are also closer to the manually corrected ones (median=15 px [0..35]) than the automatically corrected ones (median=19 px [10..37]). When differentiating novices and experts, there is a significant difference between the distances between fixations_uncorr and fixations_man for both groups ($p<0.001$), with experts having a much higher median distance (20 px [0..138]) than novices (10 px [0..25]). Also, the matches between fixations_uncorr and fixations_man on both AOI-levels are much lower for experts than for novices (e.g. 71% line-matches for novices and only 51% for experts). This suggests that the expert data contains greater errors than the novice data. This difference between the two types of expertise is probably caused by the very different recording environments and stimuli characteristics. For the most part, expert programmers were recorded at their workplaces with partly very unfavorable conditions for eye tracking, while the novices were in a computer lab. Furthermore, some of the expert stimulus programs were rather difficult in order to force the programmers to actually use their expertise, which results in longer trial durations. Expert trials have a median duration of 96 sec [57..151], much more than the 38 sec [20..69] found for novice trials. In addition, many of the novice programs are short, and unlike some of the expert stimuli hardly stretch to the edges of the screen. Thus, the expert stimuli provide more potential for error.

After correction, the percentage of matches on lines for all SCs together increases from 66% to 82%, for elements it remains identical (63%). The percentage of fixations with the AOI in scope rises from 71%

	All		NT		SC		Novices		Experts	
	uncorr	corr	uncorr	corr	uncorr	corr	uncorr	corr	uncorr	corr
median_distance	10	18	0	13	15	19	10	16	20	28
match_line	70	84	95	93	66	82	71	88	51	65
scope_line	75	88	97	96	71	87	77	91	52	75
fitness_line	89	98	98	99	87	97	90	99	78	94
match_element	68	65	95	77	63	63	70	70	45	42
scope_element	73	75	97	89	69	73	76	79	47	55
fitness_element	87	96	98	97	85	96	89	97	75	91

Table 6.7: Comparison of uncorrected and automatically corrected data with manually corrected: Median (in pixels) and percentages of fixations on the same AOI (match), fixations within scope of the same AOI (scope), and fitness, each on line- and element-level

to 87% on lines and from 69% to 73% on elements. Similarly, for both novices and experts correction increases the percentage of matches on lines and hardly changes it for elements. Despite the careful method, manual correction remains somewhat subjective and does not provide a real ground truth for comparison, hence an additional plausibility check was carried out on the corrected data.

6.4.3 Plausibility check

Two researchers, who are familiar with gaze data during programming, but were not involved in the manual correction, were asked to judge which corrected version seems more plausible - the manual or the automatic one. For each trial, the uncorrected scanpath was presented together with both corrections. The two corrected versions were shown in random order and without any indication how they were corrected. For 53% of the trials the automatic correction was marked more or equally plausible as the manual one. This circa half-and-half rating is the optimal result. If manual correction was deemed less plausible even more often, the results of manual correction had to be questioned. As conclusion, for about half of the trials, the result of the correction algorithm is more or equally plausible as the manual correction, in the other half the manual corrections were more plausible, so ultimately they are on par with each other.

6.4.4 Conclusion

Correction_nt and correction_sc were applied to a large subset of data from the EMCR study. For this data, there is no possibility to determine the correct location with absolute certainty, so the automatically corrected fixations are compared to manually corrected ones. However, while it can be assessed, how well these versions agree with each other, none can be said to be the correct one. The results show that overall, due to automatic correction much more fixations land on the same line as the manually corrected ones. However, to a certain degree, the effect varies among the subgroups of data. On element level, the two corrections differ a bit more, however for the intended analyses correctness is more crucial on line-level. In the end, automatically corrected fixations proved to be as plausible as very carefully manually corrected ones.

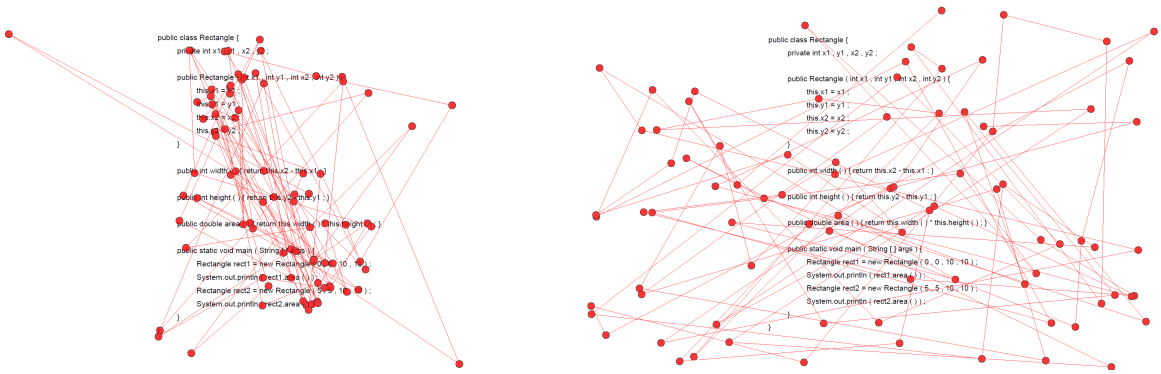
6.5 Evaluation using artificial data

As further evaluation step, correction_nt and correction_sc were applied to artificial data to verify the underlying concept and to confirm that the automatic correction neither distorts accurate fixations nor creates meaning from random data.

6.5.1 Data

In order to broaden the evaluation, other stimuli were adopted for the evaluation with artificial data than for the RLs. These stimuli should be as representative as possible. Since the NTs are highly comparable to each other, it is not of great importance which one of them is used. NT1 was already employed during the first evaluation step. From the two remaining texts, NT3 was chosen as exemplary NT stimulus, simply because it contains more words than NT2. The `Rectangle`-program (novice L3_SC1 / expert SC1) was selected as SC stimulus, since it is one of the two SCs which were presented to both novices and experts and the other of these two programs was already used for evaluation.

For every stimulus, three *ideal* sets of fixations were generated using the minimum, average, and maximum number of fixations found for that text in the EMCR data. Initially, all fixations were created on an AOI, which was chosen randomly with repetitions allowed. In order to make the data more realistic, the fixation location is not in the middle of an element, but each pixel inside an AOI has the same chance of being selected, including those at the border of an AOI. A certain degree of noise was added by scattering 10% of the created data over empty space (figure 6.15a). Such artificial fixation data could be further distorted in different ways in order to asses how well the correction method handles the resulting errors, as e.g. done by John et al. [2012, 299,300]. However, it is very difficult to judge, how realistic and representative the applied distortions are. The two previous evaluation steps are much more relevant for this information. Hence, no further alterations were put into effect. Nevertheless, in addition to the ideal fixations, *random* datasets were generated by randomly distributing fixations over the screen, again using the minimum, average, and maximum number of fixations (figure 6.15b). This approach allows to keep the stimulus and fitness table constant while testing the effect of different fixation distributions and is similar to Nüssli [2011, 61-64], who replaced actual fixation locations by random screen positions.
In total, 12 artificial datasets with 3,662 fixations were build and corrected (table 6.8).



(a) Ideal scanpath (b) Random scanpath

Figure 6.15: Exemplary artificial fixation data on SC

6.5.2 Analysis procedure and results

The NT and SC datasets were corrected with correction_nt and correction_sc respectively, including expanded AOIs and splitting the screen according to text type. The corrected versions were then compared to the original fixations, which provide an exact ground truth for comparison. As before, the Euclidean distance serves as metric for the distance. Correction results are not assessed by visualizations, e.g. plotted fitness values for the tested parameter range like done by Nüssli [2011, 61-64], since the added linear factor makes such visualizations unfeasible.

Analysis yielded the expected results (table 6.8). The six ideal, yet realistic scanpaths were not changed at all during correction, even though some of the generated fixations were on empty space and the rest was placed randomly within AOIs. Since original and corrected fixations are identical, their distance is zero in all cases and they completely match on lines and elements. The random datasets on the other hand were all modified. The original and the corrected versions still match on 86% of the lines and elements, but on average, fixations were moved 71 px away from their original position. Using

expanded AOIs as basis, the overall fitness both on lines and elements was 15% in the original random data. Correction only increased fitness to 21%, so the vast majority of fixations is still located on blank space outside of AOIs. These results corroborate that the developed approach is suitable for correcting fixation data that is mostly directed at AOIs.

Fixation set	Number of fixations	Median distance	Match-line	Match-element
nt_ideal_min	92	0	100	100
nt_ideal_mean	251	0	100	100
nt_ideal_max	648	0	100	100
sc_ideal_min	85	0	100	100
sc_ideal_mean	238	0	100	100
sc_ideal_max	517	0	100	100
nt_random_min	92	67	95	95
nt_random_mean	251	51	87	87
nt_random_max	648	81	90	90
sc_random_min	85	157	68	68
sc_random_mean	238	65	80	80
sc_random_max	517	59	85	85

Table 6.8: Summary of error correction on artificial fixations: median distance between original and corrected data (in pixels), matches between original and corrected data (in %)

6.6 Conclusion

Starting from two existing methods, a novel error correction was developed specifically for the EMCR data. It maximizes the amount of fixations on AOIs by applying a linear error function to the fixation location. As an advancement to previous approaches, fixations are corrected individually on different screen regions. In order to establish these regions, the distribution of AOIs on the stimulus is taken into account. A corrected version of the complete fixation set serves as baseline to keep the separate corrections consistent. Furthermore, AOIs at less dense areas of the stimulus are expanded to accommodate fixations with greater error. The suitability of this approach was tested using several complementary evaluation methods. The correction proved to be appropriate for the intended use and considerably improved data quality. Even though the correction was developed with a specific dataset in mind, it can be very well applied to other gaze data, as long as the prerequisite is met that gaze is mainly directed at AOIs, since the correction procedure can easily be adapted by splitting the screen to match the AOI distribution on a wide range of stimuli besides texts.

7 Analysis procedure

7.1 Overview

The purpose of the analysis is to demonstrate the use of gaze data to study code reading and to answer the two research questions. First, it will be analyzed whether there are differences between reading NT and SC and, if so, whether these differences can already be found in early novice programmers. Second, it is studied whether there are differences in how novice and expert programmers read SC.

As preparation for analysis, the stimulus texts were partitioned into areas of interest and the answers to the comprehension questions were graded. After recording the gaze with OGAMA, the raw data was exported. Only samples tracked on NT and SC stimulus texts were processed further, gaze from reading the instructions and answering the comprehension questions is not analyzed. Fixations were detected based on the adapted version of the I-DT algorithm. Subsequently the purpose-built correction procedure was performed on the fixation locations to address spatial errors. Saccades were then calculated as Euclidean distance between the corrected fixations. Finally, the data was analyzed using a comprehensive set of established as well as novel measures. Throughout the complete process, the data was inspected at various stages using both static and dynamic visualizations. Unless indicated otherwise, all steps for preparing the data and the subsequent analysis were carried out in R [R Core Team, 2019].

7.2 Areas of interest

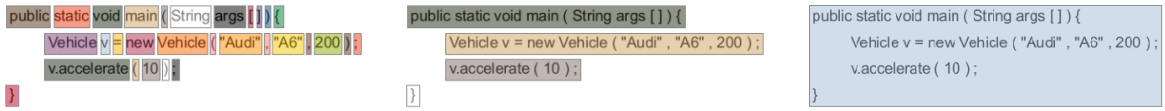
Some analyses involving gaze data can be carried out without taking the position of stimulus elements into account, e.g. studying fixation duration. Oftentimes however, it is crucial to know where certain components of the stimulus are located. The stimulus is therefore divided into AOIs, which contain the relevant parts or features [Holmqvist et al., 2011, 187,189,217]. While some stimuli, like most pictures, are fully covered by content, the EMCR stimuli consist of text surrounded by white space. The text is sectioned into AOIs using several levels of abstraction:

Elements constitute the lowest level and consist of single words in NT and code elements in SC.

Lines comprise all elements within a line of text.

Blocks combine lines into logical units. They again can be defined with different degrees of granularity, e.g. all lines belonging to a method can be subsumed into a single block or the method can be split into the two blocks ‘method-header’ and ‘method-body’, depending on the focus of the question at hand.

Figure 7.1 shows exemplary AOIs on a SC stimulus. Occasionally AOIs on different levels can coincide. If a line only contains one element, like a single closing bracket in SC, element- and line-level can be regarded as identical (see last line in figure 7.1a and figure 7.1b for an example). Likewise, a single element or line can in some contexts constitute a block-AOI by itself. Budde et al. [2017] use the concept of block-AOIs specifically for SC stimuli and developed a tool called Block Sequence Viewer for visualizing gaze data



(a) Element (b) Line (c) Block

Figure 7.1: AOIs on different levels of abstraction, from 'element' to 'block'

according to the block it was recorded on, in order to facilitate the analysis of different phases in code reading.

Alternatively to the approach described here, AOIs could be established by overlaying the stimuli with a grid and using the cells as AOIs, regardless of their content [Holmqvist et al., 2011, 212]. However with regard to the research questions, AOIs which correspond to self-contained meaningful parts of the text are more suitable. White space, i.e. areas without text, does not serve as AOI per se, but is taken into account. The vast majority of participants' gaze is directed at AOIs, though occasionally they do look on blank areas, e.g. during thinking processes [Holmqvist et al., 2011, 206,208]. For analysis measures where gaze on white space is relevant, it will be explained how the respective fixations are treated, and when analyzing which AOIs are covered by fixations, the proportion of fixations on empty spots is also provided for comparison.

The coordinates of elements and lines in the stimulus texts were obtained using eyeCode, a Python library for analyzing gaze data[1] and then processed with R. EyeCode finds AOIs by scanning the screenshot of a text stimulus for dark pixels and provides the position of each AOI together with its width and height. All resulting AOIs were visualized and inspected manually to ensure all lines and elements were encompassed correctly. Occasionally, separate elements were merged into a single AOI or one element was split into multiple AOIs, probably due to somewhat blurred areas in the screenshot. Such cases were resolved unambiguously. The obtained AOI boxes were slightly enlarged by adding up to five pixels as margin where possible. For the programs L3_SC1/SC1 and L5_SC3/SC3 block-level AOIs were constructed enclosing the `main`-methods using the coordinates of the lines and elements in `main` (figure 7.1c). A second version of all AOIs was created by expanding them to include more of the surrounding white space in less dense areas (see section 6.2.3.3 *Adapting AOIs* and figs. 6.2, 6.3, and 6.4). Only expanded AOIs are used during analysis, the original more terse versions are only needed for error correction. The AOIs are determined algorithmically, and thus very exact and objective.

After fixations were identified and their locations corrected, they were mapped to AOIs. Since all EMCR AOIs are static and have very clear-cut borders, it can be clearly decided whether the center of a fixation is located within the boundary of an AOI or not. However, the fovea allows to perceive a larger area around the fixation center. Besides, the raw gaze samples which were pooled into a fixation do spread around the fixation center, so the gaze actually hovered in the surrounding region. Thus, when identifying which AOIs were looked at, not only the AOI at the exact fixation location is counted as being seen, but all AOIs within 0.5° of visual angle.

AOI sizes vary depending on their content. Very small AOIs, e.g. operators in SC, can be problematic with regard to the eye tracker's accuracy and precision, as well as the size of the area perceived by the fovea [Holmqvist et al., 2011, 223,224]. However, margins are incorporated wherever possible and when analyzing AOI coverage, this issue is taken into account by also noting the AOIs in the fixation's vicinity. On line- and block-level, small AOIs do not pose a problem. Even though some lines are rather short, e.g. those containing only a closing bracket, they are surrounded by plenty of margin space.

Using AOIs, additional events can be identified and used for analysis. The three basic and often used AOI events are AOI hits, dwells, and transitions. A *hit* occurs when a fixation (or raw sample) lands on an AOI. The entire stay on an AOI from entry to the corresponding exit is called *dwell* and can consist of more than one fixation. Finally, moving from one AOI to another is called *transition*. Transitions can include more than one saccade, if the interjacent fixations do not hit an AOI. Leaving and re-entering the same AOI without looking at another AOI in between is usually not considered a transition, neither are movements within AOIs. Several other events can be derived from these basic events, e.g. the total skip, which occurs when a participant does not look at a certain AOI at all during the trial [Holmqvist et al., 2011, 187,189-192]. The analysis of the EMCR data is mainly concerned with hits.

[1] https://github.com/synesthesiam/eyecode, last accessed 12/05/2020

7.3 Preparation of data for statistical analysis

7.3.1 Comprehension questions

The answers to multiple choice questions were recorded with OGAMA, text answers were entered into a webpage and stored in a text file. In case of technical difficulties, answers were written down on paper. For three trials, no answer was recorded because of a software failure even though gaze data is available for the respective trial.

The correctness of the answers were graded by the author. Correct answers were scored with '1' point, partially correct answers with '0.5', wrong answers with '0' points. For the multiple choice and outcome questions grading is rather straightforward. The score for the summary question is less objective. Novices were instructed to give a summary of the SC. For experts it was additionally specified that they are supposed to describe the algorithmic idea as well as its implementation, a task that cannot be expected of very early novices. Consequently, the summaries were graded according to the participant's level of expertise.

Exemplary summaries for program L5_SC3/SC3, which were scored with a full point:

Novice: "The program requires information on producer, type and maximum speed of a car. After an increase of 10 kmh in speed the program gives the actual speed. If the maximum speed is already reached the program gives the maximum speed and does not allow any further increase in speed."

Expert: "you can create car with manufactorer, type and topspeed. increasing the current speed is possible, until top speed is reached. if its bigger than top speed, it does not increase
create a car with presets,
increas speed, but check for top speed [sic]"

7.3.2 Event detection

First, the raw data from all recordings were exported from OGAMA. Detection of oculomotor events was performed on all trials on NT and SC stimuli. After evaluating different options for event detection, a customized version of the widely used I-DT algorithm was used to identify fixations. This adapted I-DT algorithm was implemented in R especially for the EMCR data, but can be used on any raw gaze data with samples containing a timestamp together with x- and y-coordinates. After the fixation data was corrected for spatial errors, saccades were computed as Euclidean distance between fixations. Since different types of stimulus materials are associated with specific fixation durations and characteristics of the recording like sampling rate co-determine the options for event detection, different parameter combinations were assessed for the EMCR data, and a parameter set suitable for reading was adopted. All details can be found in 4 *Detecting oculomotor events*.

7.3.3 Error correction

Due to a number of factors, eye tracking data is subject to spatial errors (see 5 *Eye tracking error*). A variety of approaches for error correction were evaluated and two identified that served as basis for an correction suitable for the EMCR data. The methods were thoroughly tested with reference locations collected in a specifically designed evaluation study, with a substantial part of the EMCR data that was very carefully corrected manually, as well as with artificial fixation data (see chapter 6 *Error correction*).

Correction was performed on all 157 trials from 10 novice and 16 expert programmers. The resulting fixation locations were visualized using static plots as well as dynamic visualizations, created using R and the eyeCode library. The corrected trials were thoroughly inspected by two researchers familiar with gaze data from programmers. Only trials which were deemed usable by both reviewers were accepted for analyses which include the location of the gaze data, which resulted in 131 trials (table 7.1). The gaze data from the other 26 trials is only used to analyze fixation duration and number of fixations. Furthermore, these trials are included when studying trial duration and correctness of the comprehension task, as these measures do not depend on the quality of the gaze data.

A few trials had to undergo additional correction steps. For the NT data it was found best to not use a linear factor for correction. However, 14 of the expert trials exhibited spatial errors, for which the NT correction method was not sufficient. These trials were first scaled by computing linear factors and offsets for the whole trial, so that the majority of the fixations lie within the text area. The resulting

	Novices	Experts
NT	27	45
NT accepted after correction	25	24
SC	47	38
SC accepted after correction	47	35

Table 7.1: Number of trials in the EMCR dataset

fixation data was corrected like the other NT trials by first calculating an offset for the whole trial and then for the three sections. For six of the novice SC trials, the range of parameters for the linear factor on the y-axis had to be restricted to 1.0 to 1.1, in order to avoid unreasonable compression. For one trial the same restriction was applied to the x-axis and for another one to the x- and y-axis. Ten of the expert SC trials also had to be re-scaled before the actual correction could be applied. In six of these trials the restricted range of 1.0 to 1.1 was used for linear factors on the x- and y-axis, in three other trials on the y-axis, and in one trial on the x-axis. All parameters were determined algorithmically.

7.3.4 AOI sequences

In order to analyze sequences in the EMCR data, gaze is converted into AOI sequences (also called AOI strings), in which fixations that landed on AOIs are represented by the identifier of their associated AOI. For transforming the data into this AOI-based representation, firstly each AOI is given a unique label, usually a letter [Cristino et al., 2010, 693,694], [Day, 2010, 400], [Hansen, 2015, 65], [Holmqvist et al., 2011, 192,193,268-270], [Mathôt et al., 2012, 3], [West et al., 2006, 150,151]. However, some of the EMCR stimuli contain over 100 elements, so the alphabet does not provide enough symbols. In order to work with a larger number of AOIs, double strings can be used, i.e. each AOI is represented by two letters, e.g. “aA” [Cristino et al., 2010, 693,694] or by a letter combined with a number, e.g. “A1” [Holmqvist et al., 2011, 193]. However, sequences like “aA-aB-aC” and “A1-A2-A3” are difficult to interpret and process automatically, thus EMCR AOIs are encoded by integers representing their ordinal number in the text. Through this, processing AOI sequences is quite straightforward and more intuitive, e.g. it is immediately apparent that a transition from line 2 to line 12 equals to a jump of 10 lines, whereas when represented as movement from line B to line L the extent of the transition is less apparent.

In AOI sequences, each item represents a fixation (or dwell) that landed on the AOI of the respective number. Thus, they convey sequential as well as spatial information and coarsely represent the scanpath. Full or expanded sequences contain each fixation that hit an AOI, including those which successively landed on the same AOI. The analysis measure *reading direction* uses such full sequences, both on line- and element-level. The measures *model occurrence* and *model similarity* focus on the more overall gaze pattern, so only line-AOIs are employed and consecutive fixations on the same line will be collapsed. These compressed sequences emphasize the order in which the AOIs were visited [Day, 2010, 400], [Hansen, 2015, 65], [Holmqvist et al., 2011, 193,268-270,27], [West et al., 2006, 151]. See figure 7.2 for an example of a full and a compressed AOI sequence on line-level.

When creating AOI sequences, visits to white space have to be dealt with, e.g. the gaze in figure 7.2 moves from line 1 to empty space and back to line 1. This could be encoded as sequence 1-0-1, but since the employed sequence-based measures are only concerned with how the gaze moves among AOIs, fixations outside AOIs are not included in the AOI sequence. As long as the gaze moves without landing on white space, the movements in a full AOI sequence are equal to saccades. Movements from one AOI to another are transitions, but since full AOI sequences also include movements within AOIs, they cannot be considered a form of representation for transitions.

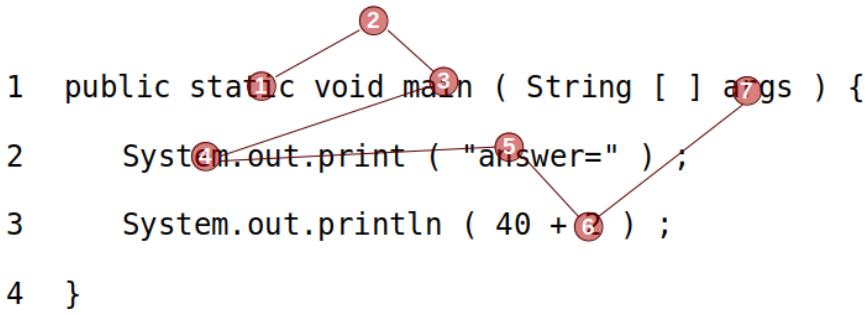


Figure 7.2: Exemplary AOI sequences on line-level

Full AOI sequence:	lines 1-1-2-2-3-1
Compressed AOI sequence:	lines 1-2-3-1

8 Analysis measures

8.1 Introduction

The two central questions of the EMCR study concern differences in reading behavior on natural language text versus source code on one hand, and between novice and expert programmers on code on the other hand. In the following, measures are presented for studying these different reading behaviors and addressing the research questions. Furthermore, the two model reading behaviors *Text Order* and *Execution Order* are specified, which serve as framework for several analyses.

The analysis measures are divided into three categories. The analysis of gaze data proceeds from single events to sequences of events. Measures that are computed from single events include fixation duration, number of fixations, saccadic amplitude, proportion of AOIs covered by fixations, and first visit to the `main`-method. Reading direction, model occurrence, and model similarity are based on sequences of growing length. Additionally, trial-related measures are employed, which are not calculated from eye movements. Trial duration and correctness of the comprehension question belonging to a trial are studied. Several of these measures are often used when working with gaze data and are not specific to a certain type of stimulus or task, e.g. fixation duration. However, some are only applicable to reading data or even exclusively to SC, e.g. first visit to `main`. Several measures were specifically devised for the EMCR data. The measures are intentionally manifold and draw on different aspects of the data, like position, duration, movement, and quantity, in order to obtain a comprehensive picture of visual code reading behavior.

Most, but not all measures will be used in answering both RQs, since some are not directly comparable across stimuli or cannot be applied to NT reading. In order to verify that the programming novices and experts are comparable NT readers and differences in how they read SC are not the result of a differing reading approach in general, the NT reading behavior of participants from both groups of expertise are compared as well. Analyses based on gaze data will be carried out on fixations and saccades respectively, not on raw data. The few measures that could potentially be calculated from raw gaze depend on the data point being at the correct location, which renders raw data unusable, as it is not corrected for spatial errors. Besides, using fixation and saccade data largely excludes artifacts, since they were filtered out during event detection.

8.2 Model behaviors

Two model reading behaviors serve as basis for studying the general reading approach and are presented in detail: *Text Order* and *Execution Order*. These models can be operationalized into prototypical sequences and compared to the actual gaze, which allows to describe and classify the observed visual behavior in terms of the model characteristics. Both models have been discussed at least partially in the first two Workshops on Eye Movements in Programming (EMIP) [Bednarik et al., 2014], [Busjahn et al., 2015b]

and the potential of this line of research was demonstrated in the pilot study by Busjahn et al. [2015a], who showed that reading Java code includes a lot of non-linear reading.

Text Order

In English, the natural language used in the EMCR study, as well as in many other European languages, it can be observed that text is generally read line by line, top-to-bottom and left-to-right, with about 10 to 15% of the saccades being directed backwards [Eysel, 2019, 763,765], [Holmqvist et al., 2011, 213,214], [Rayner, 1998, 375], [Rayner et al., 2005, 80], [Špakov et al., 2018, 2].

Text Order denotes the prototypical behavior of reading a text linearly from top to bottom. Simon [2015] proposed the term *StoryReading* for reading "code just as one might read a story in a natural language, starting at the beginning and reading through to the end" [Simon, 2015, 31]. This term was adopted by Busjahn et al. [2015a]. However, in order to emphasize that the model refers to the sequence prescribed by the text, not the somewhat noisy practice of an actual reader, which includes a certain amount of backward movement across lines and re-reading, here the term Text Order is more suitable. On source code such a linear reading can for example be observed during an initial scan [Sharif et al., 2012], [Uwano et al., 2006], [Uwano et al., 2007]. An example for Text Order is given in figure 8.1.

Execution Order

Execution Order is a model specific to SC and denotes reading a program in the order it is executed. It emerged from observing gaze data, as well as the interviews conducted with the expert programmers in the EMCR study. At the EMIP workshops, participants studied several gaze recordings of programmers reading Java codes and compiled their observations into a coding scheme [Bednarik et al., 2014, 36-41], [Busjahn et al., 2014b, 114-118], [Busjahn et al., 2014c, 5,6]. Several codes from this scheme relate and contribute to the Execution Order model:

- **JumpControl**: "Subject jumps to the next line according to execution order." [Bednarik et al., 2014, 36]
- **FlowCycle**: "The same program flow sequence is followed several times, the intent might be to gain a first understanding of the flow, strengthening and reinforcing it with repeated examinations of the same code." [Bednarik et al., 2014, 39]
- **InterproceduralControlFlow**: "The subject follows call-chains in real or simulated sequence of control flow. Intention is to understand the execution or to get the outcome of a code section. Focus is on execution between blocks." [Bednarik et al., 2014, 40]
- **IntraproceduralControlFlow**: "The subject scans lines of code in real or simulated program execution order. Intention is to understand the execution or to get the outcome of a code section. Focus is on execution on block level." [Bednarik et al., 2014, 40]

The concept of Execution Order has also been addressed in other contexts, e.g. Pennington [1987, 298-304] defines a number of abstractions of program text, and refers to information about the order in which code is executed as *control flow*. The other abstractions of program text *function*, *data flow*, and *conditionalized action* can also serve as basis for further model behaviors specific to code reading and are in part reflected in the EMIP coding scheme as well. Uwano et al. [2006, 134] and Uwano et al. [2007, 2291] state that during code reviews, reviewers tend to read the code according to control flow to simulate it. Furthermore, Xie et al. [2018] suggest teaching novices a tracing strategy which encompasses identifying the line in which the program starts executing and then following the program line-by-line using external memory tables to record method calls, variables, and so forth.

Additionally, in the EMCR study, the programming experts were interviewed after the gaze recording and asked whether they had a certain approach to read and understand the source codes. Despite their diverse backgrounds, the professional programmers describe a generally similar approach, which includes going to the `main`-method early on and to a certain degree follow the execution from there. These statements are the crucial factor for defining the overall model of Execution Order. Since almost all expert participants outline a comparable approach, the interview data serves as fundament for the Execution Order model and the measure *first visit to main*, but is not analyzed in greater detail. The experts' accounts of paying attention to a program's execution also align well with the finding of Pennington [1987, 333] that 57% of the statements in program summaries given by professional programmers were labeled as *procedural*, i.e. they referred to the program's control flow. The full interviews are in the appendix (A.4 *Expert interviews*), the following interview extracts present participants' statements regarding the

question, whether they approached the source codes in a certain way.

AE22: "Ah, there is no methodology that I always follow. I just went, go by what I feel I need, uhm, to grasp more and maybe, maybe that, that there is a clear starting point, I'll start from there, for example eyeing the code and then going to main and starting from there."

[Interviewer: "You said going from main and starting from there. What means 'from there'?"]

"Well, for example, if I see a clear path from main, where the program starts from the main and then there is some code call made from there, then I go and see what these calls are about."

BE18: [Translated from German] "Well, I tried to, quasi, follow the program flow. First, the main-method stated below as starting point, and then I tried to roughly follow the flow and then I tried to understand the details, how values are now changing."

BE26: [Translated from German] "Uhm, first get a general idea, what is actually called and in what order it is called. Like, uhm, first the public static void main here and then check, uhm, what they do."

BE29: [Translated from German] "With the first source codes, I tried to, uhm, look at the whole class, at least the complete source code and uhm, roughly look, how it is composed. Uhm, then I realized that this does not help me, so I went down to the main-method, had a look 'Ok, it calls the class, so look at the class, ah ok, the calling class is also used'. That worked better."

CO20: [Translated from German] "Uhm, first the main-method, then usually the constructor to see how parameters are passed and then the method calls. [...] I usually always looked at the main-method and after that at function call, I followed the functions."

HI27: [Translated from German] "I always tried to find the main-function and to check, which functions exist and what the general structure looks like. When which function is called by another function, what are the input and output parameters of the main function and then, what are the called sub-functions doing. Thus, a top-down approach."

IE30: [Translated from German] "So I usually try to follow the, the, the flow. So I mean, I start, I know that I myself, usually start at the top, well, have a first look, because first this is text and then I say 'Ok, aha, ok, name, class, etc.' and then I usually look at the main-method, 'What is it actually doing?'. And then I'll work from there, so that's the way it is structurally. Just having a look, what does it do, what does it call, what does that call and so on and try to use that to follow the flow."

KK24: "Not the particular approach, but I first follow the main-method and what it's trying to do. And then I looked in the method that it calls. [...]"

[Interviewer: "And then, how did you proceed from [the main]?]"

"There you can find the actual method that it's calling. And then I followed the method and then the output."

LK23: "I always would go, I always will quickly first go through the classes, over the functions and then I would immediately go to the main and to see actually what are the inputs. Actually mostly to understand what are the arrays are for and stuff. So, and then when I've seen what is in main required, then I come back to the functions and or classes and then see what there are for, and then, I mean, try to think."

MR05: [Translated from German] "Uhm, my feeling is, I start with the class name, which, if it has a descriptive name, gives a first hint, what to expect. Uhm, the first starting point here would always be the main-method, to see what will happen in general, and then successively look into the methods from there."

PA24: "Yeah, I usually start from the main program and then see what that does. Like see what gets put in to what and then after that I follow the path of the, like executing the program in my head, kind of. Seeing what you actually need to read and then lazily only reading those parts. Of course in these you will always read everything, but in most normal cases, you wouldn't need to read all the classes to understand the program, you just need the ones that are actually executed so."

SI28: "I guess, I started from the exit point. For example from the main method, then backwards, looking for the, what the, how the program is executed. Yes, something like that."

TU15: [Translated from German] "Yes, I looked roughly around at first. I looked roughly, but then I always looked for this main-method first, where it is. And so I looked a bit, what parts of the source code, I think, I looked, which parts of the source code are relevant at all, to simply say: 'ok, I do not have to read those'. But for the tasks all were relevant. However, I started with the main-method and looked a bit where the parts end up, and I also tried to work with the example, so to compute a bit mentally, instead of just looking at the algorithm."

UL29: [Translated from German] "Yes, in the sense that at first I only looked at the main-method and thus did not read from top to bottom, but rather looked first, what will be called first and then went through the program step by step."

Two participants even explicitly address not reading from top to bottom:

IE30: "Just reading the text from top to bottom is not very helpful, since it does not represent the logical sequence."

UL29: "[...] at first I only looked at the main-method and thus did not read from top to bottom [...]"

When asked, whether their approach would also be helpful for novice programmers, a number of experts explicitly recommend it. They see several advantages, especially that it allows to focus on the most relevant parts of the code and provides a course of action to work through the program.
The following statements illustrate the experts' notions how going through a program from the `main`-method is beneficial.

CO20: [Translated from German] "The advantage is that you quickly filter out unnecessary information."

Hi27: [Translated from German] "Because this is the best way to get an overview of the overall system and to only deal with the details later. And then maybe unnecessary stuff, so do not focus on unnecessary or unimportant stuff from the very first on."

LK23: "I, yeah, I think actually it's a kind of the best in a way, but I think that maybe somebody new, he will spend more time first to go through functions and then only after some time to come to main. But actually, I believe this is not. Actually I really think that the coming from main is more important and makes more sense, because the main give you real data so you actually knows what arrays supposed to be, physically. Then it's easier to read code. Cause you are more expecting, how you, what you are doing with that array or whatever, or that e.g. indexes."

PA24: "Yes. I think. It might be like time-consuming to get like the certain mentality, like what to memorize and what not to memorize when doing this. But I think, in my case I really like procedural code, because it's how the computer really runs the code, even though you try to have higher level like objects and stuff like that, but still the execution in almost everything is kind of procedural so, it, in. I think it's an easy approach for programming to get procedural stuff and then like extend that to the abstract level of like objects and stuff like that. Like, it's still kind of procedural. I think, it's easy for the brain to do stuff in sequences."

> **TU15:** [Translated from German] "And sometimes, depending on the task, it may be that many parts are superfluous and therefore reading this complete source code is not quite suitable, I would say, but maybe novices do that. I tried to see which parts are actually relevant, though. [...] For me it is certainly easier, because I can just follow a concrete example and have a look what happens and then see what is actually relevant for reading. This is especially true for larger source code examples, you just cannot read everything."

Thus, as a model reading behavior specific to code reading, it will be tested how much participants actually follow the order in which the code is executed. Figure 8.1 provides an example for Execution Order.

```
1  n = 3
2  while n > 1
3     print n
4     n = n - 1

Text Order:        1-2-3-4
Execution Order:   1-2-3-4-2-3-4-2
```

Figure 8.1: Model behaviors for program L1_SC3 (on line-level)

8.3 Single-event-based measures

8.3.1 Fixation duration

Fixations here refer to the episodes detected in raw gaze data, in which the gaze remains relatively still. Fixation duration is one of the most frequently used measures in eye tracking studies, even though it is a very individual feature and varies substantially even for the same person within a stimulus. However, when repeating a task, comparable average fixation durations can be found. Fixation duration is affected by a number of factors, e.g. text difficulty and formatting, so when a text becomes more difficult, fixation duration tends to increase. In general, longer fixations imply deeper processing and more effort. Shorter fixation durations can be found with increased experience with a task. However, expertise can also result in longer fixations. Due to a larger perceptual span, experts can extract more information per fixation and thus make use of longer, but fewer fixations [Holmqvist et al., 2011, 377-383], [Rayner, 1998, 376,392,393], [Rayner et al., 2005, 80-82].

The minimum fixation duration for the EMCR data is set to 48 ms, so fixations cannot be shorter than this. In the analysis, the distribution of fixation durations as well as the average fixation duration are studied. First, the three NTs are compared in order to establish whether they are largely comparable regarding the fixation durations they induce. To obtain a comprehensive picture of the participants customary reading behavior, fixation durations are also compared between novices and experts when reading NT. To answer RQ1, differences between fixation durations on NT versus SC stimuli are analyzed. With regard to RQ2, fixation durations from novices and experts are studied on the two stimulus programs that were read by both groups, as well as between the two programs. All available fixation durations were used for analysis, including those of the 26 trials which contain too much spatial error for mapping fixations to AOIs, since fixation location is not of interest for this measure.

8.3.2 Number of fixations

Like fixation duration, number of fixations is a widely used measure in studies using eye movements. The number of fixations on a stimulus depends on factors like its size, the difficulty of the task that is carried out, and the participant's expertise in the task. Experts usually need fewer fixations to complete a task than beginners, presumably due to having a larger visual span and skipping irrelevant parts of the stimulus, e.g. fewer fixations are reported for skilled readers than for novice readers. Furthermore, a

smaller number of fixations is associated with more efficient searching, while difficulty with a task usually results in a large number of fixations. Nevertheless, examples can be found, where experts fixate more often. Greater experience probably results in efficient information intake, so experts occasionally make more but shorter fixations than novices [Holmqvist et al., 2011, 412-414].

The number of fixations cannot be compared directly between stimuli, since they vary in difficulty and content, e.g. a text with more elements may induce a longer reading time and more fixations simply because of its length [Holmqvist et al., 2011, 225-227,412]. Thus, number of fixations will only be analyzed between participants on the same stimulus, not between stimuli. Comparisons are made between novices and experts on the three NT stimuli to obtain information about their general reading behavior. Furthermore, for RQ2, number of fixations will be analyzed for novices and experts on the two programs read by both groups. Again, fixations from all trials are used for analysis, including the 26 trials with spatial errors that cannot be used for the AOI-related analyses. Since the number of saccades is about the same as the number of fixations (see also Holmqvist et al. [2011, 403]), number of saccades is not studied.

8.3.3 Saccadic amplitude

Saccadic amplitude is computed as Euclidean distance between two fixations (see 4.3 *Event detection on the EMCR data*). Like fixation duration, saccadic amplitudes vary considerably between people on the same task, as well as over the course of a trial. For example, saccades during reading English text have an average amplitude of 2 to 3° of visual angle, but typically also contain much longer and shorter saccades, e.g. return sweeps bring the gaze from the end of one line to the beginning of the next and can be followed by a small corrective saccade, in case of undershooting. Saccadic amplitude is influenced by factors like stimulus size and task difficulty, e.g. on difficult text, saccades become shorter and are more often directed backwards [Holmqvist et al., 2011, 312-314], [Rayner, 1998, 373,375,376], [Rayner et al., 2005, 80,81].

In addition to the distribution of saccadic amplitudes and average saccadic amplitude, scanpath length is analyzed, i.e. the sum of all saccadic amplitudes [Holmqvist et al., 2011, 319]. Since saccadic amplitude is highly affected by stimulus size and location of stimulus elements, comparisons will only be carried out per stimulus, not between them. Due to different screen sizes and resolutions, the NT stimuli were partly formatted somewhat differently and are therefore not analyzed with regard to saccadic amplitude. On SC stimuli, formatting is identical, so the number of lines and elements per line remains the same for all screens and resolutions and saccadic amplitude is measured in degrees of visual angle not in pixels. Thus, saccadic amplitude is only analyzed for RQ2 to compare the reading behavior of novices and experts on SC stimuli. As saccadic amplitude is predicated on gaze positions, only trials with corrected data are used.

8.3.4 AOI coverage

AOI coverage denotes the proportion of AOIs that were looked at during a trial (see also Holmqvist et al. [2011, 421]). The measure element coverage has already been used by Busjahn et al. [2015a] and Peachock et al. [2017] to analyze SC reading, line coverage by Blascheck & Sharif [2019]. An AOI is considered as covered, if at least one fixation lands on it. However, the size of the fovea allows to see a certain area around the center of a fixation clearly and during event detection samples with some dispersion were merged into a single fixation. Thus not only the AOI at the reported fixation location is counted as being hit by a fixation, but all AOIs within 0.5° of visual angle of the fixation center. The fovea usually has an even greater span, but in order to employ an extent for which it can be assumed with great certainty that an AOI is still within foveal vision, a cautious option is chosen. This approach also depletes issues that might arise from the partly very small AOIs.

AOI coverage provides information on the spread of gaze on the stimulus and can be interpreted as an operationalization of reading depth, which has the purpose "to quantify how much of the text has been read" [Holmqvist et al., 2011, 390]. Other options for reading depth include the amount of centimeters that were read and number of fixations per word [Holmqvist et al., 2011, 390,391]. As a dispersion measure, AOI coverage is also related to the coverage, and volume under an attention map, which among a number of possible definitions, can be computed as percentage of fixated cells for a stimulus that was overlaid with a grid [Holmqvist et al., 2011, 367,368]. The implementation for the EMCR data however

does not employ equally sized AOIs, nor is the complete stimulus image divided into AOIs. Furthermore, AOI coverage is to a certain extent a counterpart to skipping proportion and skipping rate, which specify the percentage of participants that did not fixate a certain AOI [Holmqvist et al., 2011, 420].

When reading English text, most words are fixated, however skipping does occur, e.g. short words like "a" and "the" are likely to be skipped. Only about 25% of the words consisting of just two or three letters are fixated, while words with at least eight letters are almost always looked at. Furthermore, a common word has a higher chance of being skipped than an infrequent one, and words that are predictable from context are more likely to be skipped than unpredictable ones [Holmqvist et al., 2011, 420], [Just & Carpenter, 1980, 330], [Rayner, 1998, 375], [Rayner et al., 2005, 84]. However, Busjahn et al. [2014a] demonstrated that such findings from NT reading cannot be taken for granted for SC reading without verification.

AOI coverage will be analyzed both on line- and element-level. Since AOI coverage is highly depended on fixation location, only trials with corrected data are used for this analysis. Additionally to the percentage of fixated AOIs, the proportion of fixations on white space is provided. For NT, AOI coverage will be compared between stimuli as well as between the two groups of programming expertise. In order to answer RQ1, AOI coverage is analyzed for NT and lesson 1 stimuli for novices, and NT and SC for experts. With regard to RQ2, AOI coverage will be tested between novice and expert programmers on the two programs read by both groups. To allow a profound interpretation of results, it is also elicited, which lines and elements are skipped.

8.3.5 First visit to `main`

In the interviews conducted in the EMCR study, the programming experts expressed quite unequivocally that they look for the `main`-method early on when reading a program (see 8.2 *Model behaviors*). Spinellis [2003a, 20] suggest to start from `main` or a comparable entry point when reading code for the first time. Furthermore, Xie et al. [2018] found that novice programmers benefit from an explicit code tracing strategy, which includes finding the line in which the program starts executing as vital step. Therefore, in order to contrast novice and expert code reading behavior, their first visit to `main` is studied. There are the related measures *entry time in AOI* [Holmqvist et al., 2011, 437,438] and *first pass dwell time* [Holmqvist et al., 2011, 389,390], which are applicable to all sorts of AOIs. However, since the `main`-method is a SC-specific feature, the first visit to it can only be studied for RQ2.

For this purpose, it is analyzed how much time passes until a fixation lands on the `main`-method for the first time and what ordinal number the first fixation on `main` has.[1] Considering that novices might read SC slower than experts, it is also inspected, how the elapsed time till `main` relates to the whole trial duration. Fixations are accepted as being on `main`, if they hit any of the elements in the `main`-method or fall into the `main`-block (see figure 7.1c). Thus, fixations are accepted as long as they occur within the dimensions of `main`, even if they land on empty space within the block, e.g. between two lines, since the content of the `main` can still be perceived. The time till `main` is computed as duration from the beginning of the trial to onset time of the first fixation on `main`.

Additionally to when the participants arrive in `main`, their first dwell there is studied, both with regard to time and number of fixations. The time of the first dwell is computed as sum of fixation durations, so saccade durations do not contribute to the dwell time [Holmqvist et al., 2011, 387,389,390]. If the gaze leaves the `main`-block and only lingers on white space before returning to `main`, the dwell is considered as continued, but the fixations outside of `main` are not incorporated into the dwell time or the number of fixations in the dwell. However, such a disrupted dwell occurred only once.

[1] In one trial from an expert, the first two fixations happened to land on `main`. The first fixation after stimulus onset can hardly have deliberately been directed to `main` (see Holmqvist et al. [2011, 384]) and is therefore not counted as the begin of the first dwell. Close inspection of this trial showed that the second fixation was very short (84 ms) and located near the first one, so the participant was most likely still in the process of getting to the stimulus. After a brief visit to the top of the stimulus program, the person returned to `main` already with the 6th fixation. So, in this case it was decided not to use the short second fixation after stimulus onset as first visit to `main`, but the later ones starting with the 6th fixation.

8.4 Event-sequence-based measures

8.4.1 Reading direction

Based on the order of AOIs in the text, reading direction describes the proportion of gaze that moves into a certain direction. The direction is determined from pairs of AOIs in the full AOI sequence. Gaze which proceeds to an AOI that is located further back in the text than the previous one moves *forward*, movements to an earlier AOI are directed *backward*. When gaze stays on an AOI, it is *stationary*. As described in section 8.2 *Model behaviors*, English text is mostly read from top to bottom, and from left to right. In order to denote this predominant direction of NT reading, the category *linear* is included as well. It comprises gaze that follows the conventional NT reading order and either remains on an AOI or moves forward to a later one. Table 8.1 specifies the directions contained in the AOI sequence from the example in figure 7.2.

Directions:

forward:	$AOI_n > AOI_{n-1}$
stationary:	$AOI_n == AOI_{n-1}$
backward:	$AOI_n < AOI_{n-1}$
linear:	$AOI_n \geq AOI_{n-1}$

Start AOI	End AOI	Direction
line 1	line 1	stationary / linear
line 1	line 2	forward / linear
line 2	line 2	stationary / linear
line 2	line 3	forward / linear
line 3	line 1	backward

Table 8.1: Gaze directions for the AOI sequence 1-1-2-2-3-1 from figure 7.2: 40% of the gaze moves forward, 40% remains stationary, 20% moves backward, and 80% follows the linear reading direction.

A similar approach to capture the direction of gaze was described by Busjahn et al. [2015a] and put to use by Peachock et al. [2017] and Spinelli et al. [2018]. Several types of saccades were differentiated, mostly on basis of their horizontal and vertical direction. In order to study the prevailing reading direction in the EMCR data, a more general approach is chosen, which is based on the sequence of visited AOIs rather than on saccades. If the gaze moves among AOIs without in-between fixations on white space, the movements correspond to saccades. This is the case most of the time, but occasionally fixations do land on blank space and are not part of the AOI sequence. 9.3.1 *Reading direction* provides information on the extent of movements with intermediate fixations outside an AOI to allow for an estimate of how much the AOI sequences are build up of saccades. Furthermore, forward movements are not distinguished according to whether the gaze went to the next AOI or a later one, and analyzing direction on line- and element-level replaces the previous differentiation into vertical and horizontal movements.

While reading direction specifies the percentage of gaze that moves into certain predefined directions, other directional measures often give the direction in degrees, e.g. saccadic direction [Holmqvist et al., 2011, 301-303]. As approach to quantify directions over the whole trial, reading direction is to some extent connected to scanpath direction [Holmqvist et al., 2011, 310,311]. Additionally, the proportion of backward gaze largely corresponds to regression rate [Holmqvist et al., 2011, 426]. However, reading direction is a more abstract measure than e.g. saccadic direction, since the direction categories were derived from the order of AOIs in the stimulus. The orientation of gaze with regard to the AOIs is of central interest, not the angular direction of physical eye movements.

During analysis, only the proportion of linear reading is used for statistical testing, since the main focus lies on the impact of the customary NT reading behavior, but the proportions of the other directions are reported to complete the picture of reading behavior. Reading direction will be analyzed on line- as well as on element-level. Since some elements form rather small AOIs, reading direction on element-level is slightly more susceptible to noise than other measures. Only trials with corrected data are used for

analysis, since reading direction is strongly dependent on fixation location. For NT, reading direction will be compared between stimuli, and between novices and experts. With regard to RQ1, reading direction is analyzed for NT and SC from lesson 1 for novices, and NT and SC for experts. To answer RQ2, reading direction will be studied on the two programs read by both groups of expertise and between the two SC stimuli.

8.4.2 Model occurrence and model similarity

The whole scanpath is analyzed to assess how much the overall reading behavior resembles the two models Text Order and Execution Order. Based on these models, ideal AOI sequences are constructed for each stimulus and compared to the gaze recorded from participants. This approach allows to characterize the visual behavior with regard to the model, e.g. a gaze sequence with a high similarity to the Text Order model can be regarded as a rather linear reading approach. Since the general reading behavior is of interest, collapsed AOI sequences on line-level are used throughout for model occurrence and similarity.

Usually, scanpath comparison is applied to determine how similar actual scanpaths are [Holmqvist et al., 2011, 273,274,346], however model and actual scanpaths can be compared just as well. From instruments for scanpath comparison that are suitable for the intended analysis, two are of most interest: string edit distance and alignments. The *string edit distance* (also called Levenshtein distance [Levenshtein, 1966]) of two strings is the smallest number of insertions, deletions, and substitutions needed to convert one string into the other. The more similar the strings are, the smaller their distance. However, even though several modifications of the string edit distance exit, it is not very flexible and when used for gaze data very susceptible to noise [Anderson et al., 2015], [Cristino et al., 2010], [Dolezalova & Popelka, 2016, 2,3], [Hansen, 2015, 65,66,79-81,115], [Holmqvist et al., 2011, 348-353], [Kruskal, 1983, 215-219], [West et al., 2006, 150,151]. For example, Hansen [2015, 65,66,79-81,115] used the string edit distance to compare the scanpaths of programmers looking at the same SC. It turned out to not be an adequate analysis tool, as even for simple stimulus programs the scanpaths differed too much between participants to yield meaningful results regarding their distance. It is assumed that this is due to the rather open nature of the experiment's task and the absence of a time constraint. Since both conditions are also present in the EMCR study, the string edit distance is not an optimal choice.

Sequence alignment is a related technique to assess the similarity of sequences and also has been adopted for analyzing eye tracking data before. West et al. [2006] introduce eyePatterns, a tool for comparing gaze sequences using the string edit distance as well as global sequence alignment, and outline how alignments can effectively be applied for detecting similarities and patterns in scanpaths, and for clustering them. Cristino et al. [2010] present ScanMatch, an analysis approach using global alignments. In order to compare ScanMatch to the string edit distance, artificial sequences were created and gradually distorted. ScanMatch classified the sequences considerably better than the string edit distance, even for higher levels of noise, whereas the performance of the string edit distance already deteriorated when slight noise was added. Two further experiments were conducted with actual gaze sequences to prove that global alignments are a stable and reliable method for scanpath comparison. Additionally, ScanMatch demonstrates the flexibility of alignments, since besides sequence, it allows to take characteristics of the AOIs as well as fixation duration into account when calculating similarity. Anderson et al. [2015] compared several scanpath comparison methods based on the established prediction that the scanpaths of one person seeing the same image twice are more similar than the scanpaths of different people viewing the same image. Even though both string edit distance and ScanMatch found this effect, ScanMatch had a considerably higher effect size and overall performed very well for all studied conditions. Day [2010] studied the validity of a global alignment algorithm for characterizing decision strategies based on gaze. As expected, search behaviors based on the same strategy received higher similarity scores than those from different strategies. Furthermore, a classification system based on similarity scores predicted the underlying strategy significantly better than chance. Dolezalova & Popelka [2016] developed ScanGraph, a tool for scanpath analysis that also employs the string edit distance as well as global alignments. They propose a refined way for detecting similarities based on the obtained string edit distance or alignment score, e.g. by accounting for varying scanpath lengths and including only the similar sequences when visualizing the detected clusters. Finally, in a feasibility study with programmers reading Java, Busjahn et al. [2015a] established that global alignment can be applied to study the similarity of gaze to model sequences and the methodology and findings of this study contributed to the development of the measures used to analyze sequential aspects of the EMCR data. The alignment procedure from Busjahn et al.

[2015a] was also employed by Peachock et al. [2017].

Consequently, alignments will be used for assessing the similarity of the EMCR data to the model behaviors. Some issues described for the string edit distance can potentially also arise in alignments, e.g. fixations that are located close to each other but on different AOIs can cause disproportionately small similarity values [Anderson et al., 2015, 1378], [Holmqvist et al., 2011, 351-353]. However, only AOIs on line-level and collapsed sequences are considered for analyzing the overall reading approach, thus such small-scale differences are of hardly any consequence.

8.4.2.1 Sequence alignment

Alignments are used to determine the similarity of sequences and to what extent a given sequence is part of another one. They are often applied in bioinformatics to compare biological sequences like DNA in order to identify common genes and ancestry. In order to align sequences so that they match at as many positions as possible, gaps (blank characters) can be introduced in either sequence, e.g. when one sequence is missing an item, which is present in the other sequence at the corresponding position. Alignments are graphically represented by stacking the sequences on top of each other to make the correspondence between their items visually evident [Böckenhauer & Bongartz, 2003, 78-80], [Cristino et al., 2010, 693], [Haque et al., 2009, 96], [Holmqvist et al., 2011, 274,275], [Kruskal, 1983, 202,203,207,208,211,226-230], [Pazos & Chagoyen, 2015, 10,11], [Zimmermann, 2003, 47,49]. Figure 8.2 provides an example.

```
2 1 3 – – 2 – – – 3 – – – – 4 1 3 3
| | |     |       |         |   | |
2 1 3 2 1 2 4 4 3 3 1 4 4 2 4 2 3 3
```

Figure 8.2: Example of an alignment: Matching items are indicated by '|', gaps by '–'.

Depending on the number of sequences involved, the approaches are classified as *pairwise* (or binary) and *multiple* sequence alignment [Haque et al., 2009, 97], [Pazos & Chagoyen, 2015, 10,20], [Zimmermann, 2003, 48,54]. With regard to gaze sequences, multiple sequence alignment can be used to cluster sequences and determine sequences that belong to certain groups, e.g. novices and experts. However, since here gaze sequences are only compared to the model, not to each other, multiple sequence alignment is not in the scope of this work and only pairwise alignments are employed for analyzing the EMCR data. Due to the huge number of possible combinations, sequence alignment is computationally intensive and complexity increases exponentially with the size of the sequences. There are both optimal and heuristic algorithms for this type of problem. *Optimal algorithms* find the best alignment, while *heuristic algorithms* only provide a sub-optimal alignment, but are faster. Heuristic algorithms were largely developed for biological sequences, which can be very long and have an enormous amount of possible combinations [Haque et al., 2009, 96-98], [Pazos & Chagoyen, 2015, 15-20], [Zimmermann, 2003, 67-70]. Since there are only few comparatively short sequences that have to be aligned for analyzing the EMCR data, only optimal methods will be considered. Alignments can be adjusted manually [Pazos & Chagoyen, 2015, 23], however as with the other steps of data processing, an automatic approach is preferred to ensure objectivity.

When aligning two sequences, at first a score is defined for pairs of items, then the sum of all partial scores is maximized in order to get the final alignment score. The relationship between pairs of items in an alignment can be classified into four types:

- **Match**: pairing of two identical non-blank items
- **Mismatch** (also called substitution): pairing of two different non-blank items
- **Insertion**: pairing of a gap in the first sequence and a non-blank item in the second
- **Deletion**: pairing of a non-blank item in the first sequence and gap in the second

Examples for the four types can be seen in table 8.2:
Instead of differentiating between insertions and deletions, columns that contain a gap in either sequence can be subsumed into a single type called indel. However, depending on the focus of the analysis, these cases can be treated differently and are therefore differentiated. Pairings of matching items contribute positively to the score, while mismatches and pairings of a non-blank item with a gap usually contribute negatively [Böckenhauer & Bongartz, 2003, 79,80], [Cristino et al., 2010, 694], [Haque et al., 2009, 97],

Match	Mismatch	Insertion	Deletion
1	1	-	1
1	2	1	-

Table 8.2: Types of pairings

[Holmqvist et al., 2011, 274,275], [Kruskal, 1983, 207-209,211], [Pazos & Chagoyen, 2015, 11], [Zimmermann, 2003, 50-54].

8.4.2.2 Types of pairwise alignments

Pairwise sequence alignment algorithms can be further classified into global, local, and glocal (see table 8.3). A global alignment determines the degree of overall similarity between two sequences, while a local alignment is useful when the similarity is restricted to particular regions. A glocal alignment is mostly equivalent to a global one, but is especially designed for cases, in which the lengths of the two sequences differ substantially. It fits the shorter sequence into the longer one. If the two sequences are very similar, all three approaches yield highly comparable results [Böckenhauer & Bongartz, 2003, 81,87,89], [Haque et al., 2009, 97], [Holmqvist et al., 2011, 274-276], [Pazos & Chagoyen, 2015, 12,13], [Zimmermann, 2003, 56].

Global alignment
The Needleman-Wunsch algorithm [Needleman & Wunsch, 1970] can be used to find an optimal global alignment for two sequences in their entirety and calculate their overall similarity. It employs the paradigm of dynamic programming, so the optimal alignment of prefixes is used to create the best overall alignment. The algorithm consists of a forward and backward iteration procedure. First, a matrix with all scoring possibilities is computed. For each cell, it is evaluated which pairing results in the highest score at this point – aligning the two current items or aligning one of the items with a gap. Then the optimal decisions made at each step are traced back through the matrix in order to build the alignment [Böckenhauer & Bongartz, 2003, 81-87], [Cristino et al., 2010, 694], [Day, 2010, 398,399], [Haque et al., 2009, 97,98], [Holmqvist et al., 2011, 275-277], [Kruskal, 1983, 220-225], [Pazos & Chagoyen, 2015, 12], [West et al., 2006, 151], [Zimmermann, 2003, 56,59].

Local alignment
For optimal local alignments, the Smith-Waterman algorithm [Smith & Waterman, 1981] finds highly similar sub-sequences within two given sequences. It is useful when sequences have a low overall similarity, but contain similar parts. The procedure is comparable to the Needleman-Wunsch algorithm and uses dynamic programming. A matrix is used to align the prefixes of both sequences and a trace back procedure is employed to construct the alignment. The main difference to global alignment is that in the scoring matrix, negative scores of aligned prefixes are set to zero. Thus, all sub-sequences have the same chance of achieving a high score without being afflicted by their prefixes [Böckenhauer & Bongartz, 2003, 87-89], [Haque et al., 2009, 97,98], [Pazos & Chagoyen, 2015, 12], [West et al., 2006, 153], [Zimmermann, 2003, 62].

Glocal alignment
A glocal (also called global-local) approach finds the sub-sequence of the longer sequence which is most similar to the shorter sequence. The optimal glocal alignment algorithm is closely related to the Needleman-Wunsch algorithm. It applies dynamic programming and uses a similar forward and backward iteration. However, at the beginning and/or at the end of the longer sequence, gaps can be inserted without a penalty, so that the shorter sequence can start or end at any position of the longer one. There are several variants of glocal alignment, depending on whether gaps are only penalty-free at the beginning, only at the end, or both [Böckenhauer & Bongartz, 2003, 89-91], [Zimmermann, 2003, 60,61].

Type	Alignment	Score
global	– – – – – 1 2 – 3 – – 4 – – 	| | | | | 2 1 3 2 1 2 4 3 1 2 4 2 3	-5
local	2 3 | | 2 3	2
glocal	– – – – – – – – – 1 2 3 4 – – 	| | | 2 1 3 2 1 2 4 3 1 2 – 4 2 3	2

Table 8.3: Types of pairwise alignments. The exemplary sequences are 1-2-3-4 and 2-1-3-2-1-2-4-3-1-2-4-2-3. Scoring: match = 1, mismatch, insertion, and deletion = -1

8.4.2.3 Adaption for EMCR data

In order to assess how well the models Text Order and Execution Order are reflected in the gaze sequences, two types of measures are employed. *Model occurrence* expresses how much the complete model is present at some point in the gaze. By means of glocal alignment, the model sequence is aligned to the gaze. To obtain the most similar overlap of model and gaze, gaps that occur prior to the begin of the model or after its end are not counted. Local alignments are not suitable for this measure, as they find the most similar sub-sequence of model and gaze, but it is of interest how much the whole model exists within the gaze, not just if a part of it is a good fit (see table 8.3). Finding highly similar sections is also a task worth studying, but out of scope of this work.

Global alignments are used to measure *model similarity*, the overall similarity of model and gaze. This type of alignment is very strict and reading a text more than once results in a very low similarity score, even if each round of reading strongly follows the model. Therefore two variants of global alignments are implemented. *Naive* global alignment is based on the somewhat naive assumption that the stimulus is read only once and tests how similar the gaze is to the model as it is. In *dynamic* global alignment, the model can be repeated to account for re-reading. First, the gaze is aligned with only one instance of the model. In the next step, two model instances are concatenated and aligned with the gaze, then tree instances and so forth. The maximum number of model repetitions to try depends on the ratio of model and gaze length. It is calculated by first dividing the length of the gaze sequence by the length of the model. If the division has a remainder other than zero, one repetition is added, in case of the gaze sequence being shorter than the model. Finally the number of repetitions is doubled to ensure that all potential instances of re-reading are captured (equations 8.1 and 8.2). The alignment that resulted in the highest score is used and stored together with the number of model repetitions needed to produce it. The naive and dynamic approaches were presented in similar form by Busjahn et al. [2015a, 258,259]. Table 8.4 illustrates the three approaches that will be used for analysis.

$$\text{quotient\_sequences} = \text{length(gaze)} \div \text{length(model)} \tag{8.1}$$

$$\text{max\_repetitions} = \begin{cases} (\text{quotient\_sequences} + 1) * 2 & \text{if (length(gaze) mod length(model))} > 0 \\ \text{quotient\_sequences} * 2 & \text{else} \end{cases} \tag{8.2}$$

A central aspect of sequence alignment is to chose a suitable scoring scheme, i.e. parameters for matches, mismatches, insertions, and deletions. The scoring strongly influences the resulting alignment [Day, 2010, 402]. Using '0' for matches and '1' otherwise effectively counts insertions, deletions, and substitutions and thus calculates the string edit distance [Böckenhauer & Bongartz, 2003, 81], [Holmqvist

Type	Measure	Alignment	Score
glocal	Model occurrence	– – – – – – – – – 1 2 3 4 – – | | | 2 1 3 2 1 2 4 3 1 2 – 4 2 3	2
naive global	Model similarity	– – – – 1 2 – 3 – – 4 – – | | | | 2 1 3 2 1 2 4 3 1 2 4 2 3	-5
dynamic global	Model similarity	– 1 2 3 4 **1 2** – **3 4** 1 2 3 4 **1 2 3 4** | | | | | | | | | 2 1 – 3 2 1 2 4 3 – 1 2 – 4 – 2 3 –	2

Table 8.4: Types of alignments used for analysis. The exemplary sequences are 1-2-3-4 and 2-1-3-2-1-2-4-3-1-2-4-2-3. In the dynamic global alignment repeating the model four times resulted in the highest similarity between model and gaze, the maximum number of model repetitions that was tested is eight. Scoring: match = 1, mismatch, insertion, and deletion = -1

et al., 2011, 276]. When comparing biological sequences, the score values for aligned pairs of items are usually obtained from scoring (or substitution) matrices, which are constructed empirically from observations on substitution frequencies [Böckenhauer & Bongartz, 2003, 92-95], [Haque et al., 2009, 97], [Pazos & Chagoyen, 2015, 11], [Zimmermann, 2003, 52-54]. Day [2010, 400-402] uses Monte-Carlo simulations to test different sets of parameters for a scoring that results in an optimal accuracy when predicting the strategy associated with the gaze. However, the EMCR data does not contain any valid associations that allow such a simulation. Even using the gaze during NT reading for finding a scoring that results in high similarity scores for Text Order is not reliable as it presumes that the NTs are read according to Text Order, which cannot be taken for granted. Consequently the scoring scheme for the EMCR data has to be derived theoretically. Based on the consideration that the model sequence represents the ideal, while the gaze is a sequence with mutations, Busjahn et al. [2015a, 258] implemented a higher penalty for gaps in the model than in the gaze sequence. Their scoring is '3' for a match, '-3' for a mismatch, '-2' for a gap in the model and '-1' for a gap in the gaze. When analyzing the EMCR data, a more concise and symmetric scoring will be used. With regard to the model it amounts to the same, whether a line from the model was skipped, another line was read instead, or a line is read in addition to the model. For the measures model occurrence and model similarity all such cases are the opposite of a match and therefore scored equally. If the line read by a participant matches the line in the model, the pairing is scored with '1'. Mismatches, insertions, and deletions get the symmetric opposite score of '-1'. Usually, the gap penalty provides the threshold for which two items should not not be aligned but, instead, a gap created. If a mismatch results in a lower score than creating a gap, the gap will be favored [Cristino et al., 2010, 694]. However, here gaps and mismatches are equally negative.

Alignments allow a number of additional adjustments to the scoring, e.g. Cristino et al. [2010] demonstrate that a substitution matrix can be used to encode how close AOIs are in space or whether they have a semantic relationship, so looking at line 2 instead of line 1 could get a better score than looking at line 8. Furthermore, gap penalties can be differentiated depending on whether the gap is the first or a subsequent one in a block of gaps (affine gap scoring). This is mostly relevant for long contiguous blocks of gaps. For analyzing the EMCR data a different modification is introduced to hone the scoring scheme. Some lines in the Java programs are of minor importance, e.g. lines that only contain a closing bracket. If such a line occurs in the model, but is not looked at by the participant, that is less critical than skipping a line containing an actual computation, since the single bracket can be perceived in the periphery without being directly fixated. Besides, participants were instructed that all programs compile

without error, so they could expect them to be there. Such skippable AOIs get a reduced gap penalty of '-0.5'. It applies when a skippable line in the model is paired with a gap in the gaze and when a participant reads a skippable line even though the model does not include it. In the EMCR analysis only lines which contain nothing but a bracket are declared as skippable, though for other analyses different AOIs can very well be viewed as such. The final scoring scheme is presented in table 8.5.

Pairing	Score
Match	1
Mismatch	-1
Insertion	$\begin{cases} -0.5 & \text{if AOI is skippable} \\ -1 & \text{else} \end{cases}$
Deletion	$\begin{cases} -0.5 & \text{if AOI is skippable} \\ -1 & \text{else} \end{cases}$

Table 8.5: Scoring scheme used for EMCR data

Alignment scores also depend on the length of the two sequences. Aligning two long identical sequences will result in a higher score than aligning two short identical sequences, even though they are just as similar [Cristino et al., 2010, 695], [Dolezalova & Popelka, 2016, 7]. Table 8.6 gives an example of a short global alignment between two quite similar sequences that receives the same similarity score as the alignment of two longer, but more dissimilar sequences. In the longer alignment, the greater number of matches compensates for the additional gaps. The lengths of Text and Execution Order sequences differ within and between stimuli and the lengths of the gaze sequences also are highly variable. Thus, in order to make the alignment scores comparable for analysis, they have to be normalized.[2]

Alignment	Score	Model length	Normalized score
− 1 2 3 4 − | | | | 2 1 2 3 4 3	2	4	0.5
− 1 2 − 3 4 5 − − 6 | | | | | | 2 1 2 6 3 4 5 3 2 6	2	6	0.3

Table 8.6: Example for normalization in a global alignment, scoring: match = 1, mismatch, insertion, and deletion = -1

Day [2010, 399] uses the number of matches in the alignment divided by the length of the alignment as similarity score (equation 8.3). This approach corresponds to the percentage of exactly matching residues in biological sequences [Pazos & Chagoyen, 2015, 11,12].

$$\text{normalized\_score}_{\text{ Day}} = \frac{\text{number of matches}}{\text{length(alignment)}} \tag{8.3}$$

Cristino et al. [2010, 695] propose to normalize the alignment score by the product of the highest possible score from the substitution matrix and the length of the longest sequence (equation 8.4):

$$\text{normalized\_score}_{\text{ Cristino et al.}} = \frac{\text{score}}{\max(\text{substitution matrix}) * \text{length(longest sequence)}} \tag{8.4}$$

[2]Thanks to Jitka Dolezalova and Stanislav Popelka for the useful discussion about normalization.

Dolezalova & Popelka [2016, 7] divide the obtained alignment score by the length of the longer sequence, since it represents the greatest similarity possible for two sequences (8.5).

$$\text{normalized\_score}_{\text{ Dolezalova\_Popelka}} = \frac{\text{score}}{\text{length(longest sequence)}} \tag{8.5}$$

For biological sequences a number of further forms of normalization exist [Haque et al., 2009, 97], [Pazos & Chagoyen, 2015, 11,12], [Zimmermann, 2003, 55,56], but are not applicable to the EMCR data.
The presented approaches were devised for comparing two sequences of which none takes prevalence over the other. They do not fully capture the concept of defining similarity with regard to a model. A normalization that is based on the length of the alignment or the length of the gaze (which is usually the longer sequence) veils the relation between alignment score and the given model. Thus, the determining factor for making alignment scores comparable between different models, stimuli, and participants when analyzing the EMCR data is actually the length of the model (see equation 8.6). Using this normalization, the best possible match between two sequences will result in a normalized alignment score of '1'. Table 8.6 provides examples for normalized scores.

$$\text{normalized\_score} = \frac{\text{score}}{\text{length(model)}} \tag{8.6}$$

In order to allow for a thorough interpretation of the alignment results, it is relevant to determine how linear the Execution Order of each stimulus is. For this purpose, Text and Execution Order sequences of every text are aligned using the naive global alignment approach and the same scoring scheme as for the gaze sequences (table 8.5), including the reduced penalty for lines containing only a closing bracket (table 8.7). The order in which the lines are executed was obtained from Jeliot [Moreno et al., 2004].[3] For the three novice programs in lesson 1, as well as for L5_SC3/SC3 (`Vehicle`-program), both model sequences are rather similar to each other. The Execution Order of these programs can be regarded as predominantly linear.

	L1_SC1	**L1_SC2**	**L1_SC3**	**L3_SC1/SC1**	**SC2**	**L5_SC3/SC3**
Length Text Order	6	6	4	18	22	22
Length Execution Order	3	9	8	31	52	17
Score	1	0	0	-9	-35	4
Score normalized	0.2	0	0	-0.5	-1.6	0.2

Table 8.7: Naive global alignment scores between Text and Execution Order sequences per SC stimulus

In the analysis, it is first evaluated how much the Text Order model fits to the gaze when reading NT, both with regard to the three different stimuli and level of programming expertise of the participants. For RQ1, the average scores from aligning the gaze to Text Order sequences are compared for NT and SC stimuli. Additionally, for SC stimuli the scores for Text and Execution Order are compared. For the second RQ, Text and Execution Order scores are compared between novices and experts, as well as between the two stimuli read by both groups of expertise. Since model occurrence and model similarity focus on the overall reading behavior only collapsed gaze sequences are used for this analysis.

[3]Thanks to Andrés Moreno for tweaking the Jeliot output to provide the respective lines.

8.5 Trial-based measures

In addition to measures based on gaze data, two measures are analyzed that refer to the trial. A trial starts the moment the stimulus text is presented and ends when the participant clicks to proceed to the comprehension question.

8.5.1 Trial duration

Trial duration denotes the time span from the beginning of a trial till its end. Raw timestamps are used to establish a trial's start and end point. Participants presumably started reading the texts right from the moment they were presented, so trial duration largely overlaps with the time participants needed to read and understand the text sufficiently to decide to proceed to answering the comprehension question.

All available trial durations are used for analysis, including the 26 trials in which the gaze was not mapped to AOIs due to insufficient data quality, since trial duration is not influenced by spatial errors in the recording. In order to test whether the NT stimuli are comparable with regard to the time needed to read them, the trial durations for all three NTs are compared to each other. Furthermore, in order to establish that the participants of the novice and expert programmer groups do not differ in their ability to read NT, the trial durations of both groups are tested for differences. Comparing trial durations between NT and SC does not yield meaningful results regarding the first research question. However, for the second question, trial durations of the two SCs that were read by both novices and experts can be compared between the two groups, since programming experts are expected to understand the SCs faster than novices. Finally, trial durations are compared between the two SCs. They are similar in length, so longer trial durations for one program would suggest a higher difficulty.

8.5.2 Correctness of comprehension question

Following each text stimulus, participants were asked a comprehension question. Correct answers were scored with '1', partially correct answers with '0.5', and incorrect answers with '0' points (see 7.3.1 *Comprehension questions*). In order to establish that there is neither a difference between the NTs, nor between novices and experts with regard to reading NT, correctness scores are compared between the NT stimuli and per NT between the two expertise groups. Again, comparing NT and SC does not provide meaningful outcomes, but for the second research question comprehension scores from novices and experts on the two SCs read by both groups are tested for differences. Additionally, the scores by all participants between the two programs are studied to check whether one program was more difficult to comprehend than the other. Since the comprehension score is not dependent on the gaze recording, all available scores are used for analysis, including those trials of which the corresponding gaze data has low quality or is missing because of a software failure.

8.6 Summary of analyses measures

In order to demonstrate the potential of using gaze data to study code reading, a number of measures were selected or newly devised to contrast NT and SC reading as well as novice and expert SC reading behavior. While there are more events, measures, and levels of abstraction that could be used to analyze the EMCR data and code reading in general, the analysis focuses on the most promising ones. Table 8.8 provides an overview of the presented analysis measures.

Measure	Basis	AOI level	RQ
Fixation duration	single event	-	1,2
Number of fixations		-	2
Saccadic amplitude		-	2
AOI coverage		line, element	1,2
First visit to main		block	2
Reading direction	event sequence	line, element	1,2
Model occurrence and model similarity		line	1,2
Trial duration	trial	-	2
Correctness of comprehension question		-	2

Table 8.8: Overview of analysis measures

9 Analysis results

9.1 Introduction

The results are presented per analysis measure. In total, gaze data from 157 trials of 10 novice and 16 expert programmers was analyzed. In order to test for normality, Kolmogorov-Smirnov tests to compare the data with a normal distribution were combined with a histogram plot. In case of an inconclusive outcome, a Shapiro-Wilk test was applied additionally. The value of 0.05 serves as level of significance. When necessary, results were corrected for multiple testing using the Benjamini-Hochberg procedure.

For research question 1, data from reading NT is compared to data from reading SC. Since for novices the focus is on early SC reading, only the three programs from the first lesson are used for comparison with the three NTs. For experts, the NTs are contrasted to all three SCs. For one novice and three experts the data quality on all NT stimuli was too low for analyses employing fixation location. Consequently, for such measures these four participants had to be excluded as no NT data is available for comparison. For research question 2, all available trials on the two programs read by both novices and experts are included.

9.2 Single-event-based measures

9.2.1 Fixation duration

First, fixation durations on the three NT stimuli were analyzed. Overall, there are 17,649 NT fixations from 72 trials and 26 participants (10 novices, 16 experts). Fixation duration is skewed to the right, but median fixation durations per trial are normally distributed. The observed durations range from 49 to 3,229 ms, with a median of 183 ms [100..267]. A two-way ANOVA revealed no difference between the median fixation durations of the three texts ($p=0.807$), but between the novice and expert participants ($p=0.001$). Novices have a higher mean median fixation duration on NT (219 ms, SD=41) than experts (175 ms, SD=47)[1], see tables 9.1 and 9.3, as well as figures 9.1a and 9.2a. The differences between novices and experts persist over the three texts (figure 9.2b).

Kolmogorov-Smirnov tests between the three NT stimuli show that the distribution of fixation durations is comparable across stimuli (table 9.2 and figure 9.1b). In order to compare the fixation durations on NT between novices and experts, all NT fixations per participant were accumulated. A Kolmogorov-Smirnov test between fixation duration distributions showed a significant difference between the two groups of expertise ($p<0.001$), see figures 9.2c-d. All p-values were corrected for multiple testing.

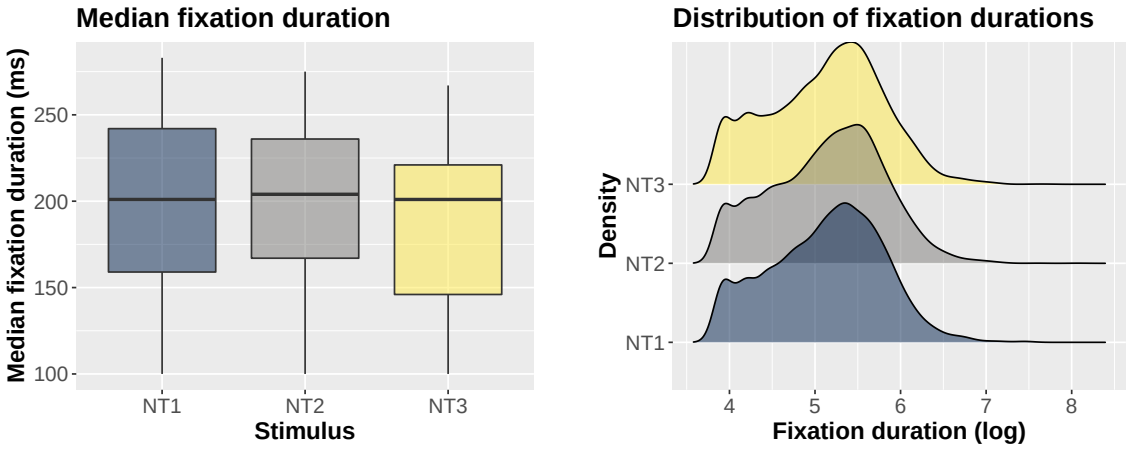
[1] Mean values are given together with standard deviation (SD).

	NT1	**NT2**	**NT3**
Number of trials - novices	10	9	8
Number of trials - experts	15	15	15
Number of fixations - novices	2,109	2,159	1,742
Number of fixations - experts	3,357	4,269	4,013
Mean median fixation duration - stimulus	196 ms (SD=53)	197 ms (SD=53)	189 ms (SD=48)
Mean median fixation duration - novices	223 ms (SD=53)	220 ms (SD=35)	222 ms (SD=34)
Mean median fixation duration - experts	179 ms (SD=47)	183 ms (SD=57)	172 ms (SD=46)

Table 9.1: Fixation durations on NT stimuli

Comparison	**NT1-NT2**	**NT1-NT3**	**NT2-NT3**
p-value	0.156	0.563	0.414

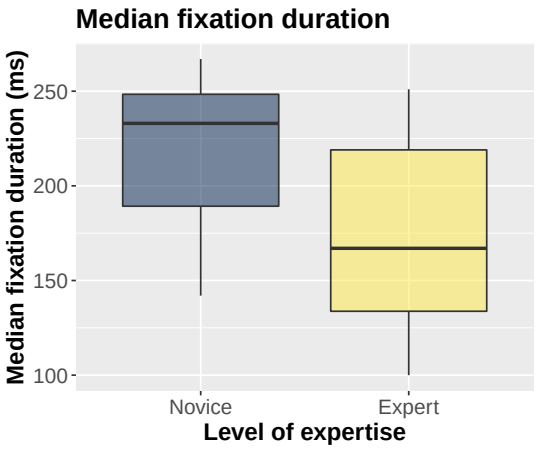
Table 9.2: Results of the Kolmogorov-Smirnov tests between the distributions of fixation duration on NT stimuli, p-values have been corrected for multiple testing



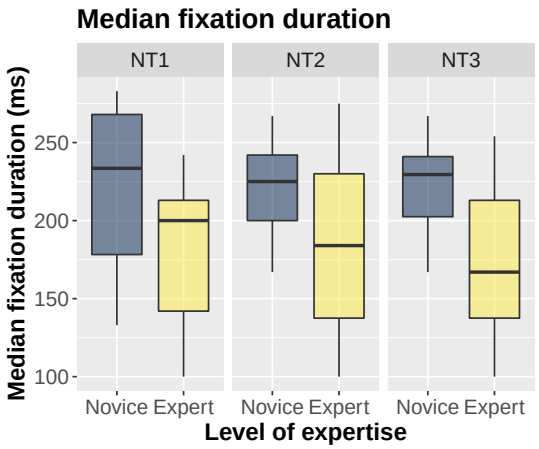
(a) Median fixation durations per text

(b) Distribution of fixation durations per text

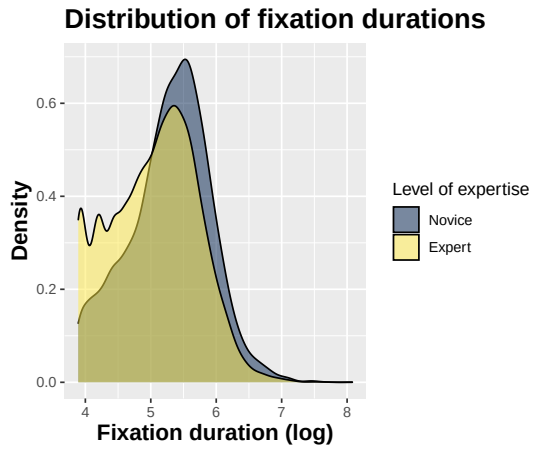
Figure 9.1: Fixation durations on NT stimuli



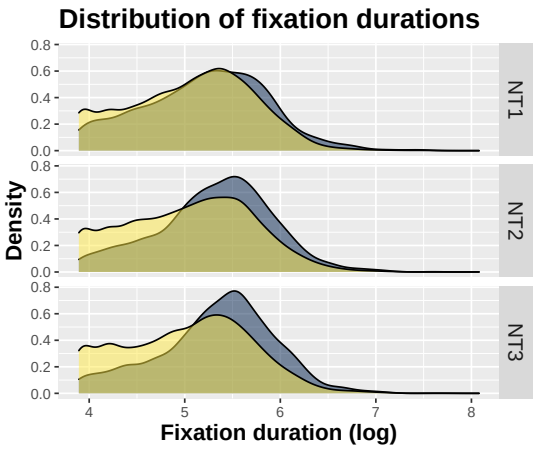
(a) Median fixation durations over all NTs



(b) Median fixation durations per NT



(c) Distribution of fixation durations over NT



(d) Distribution of fixation durations per NT

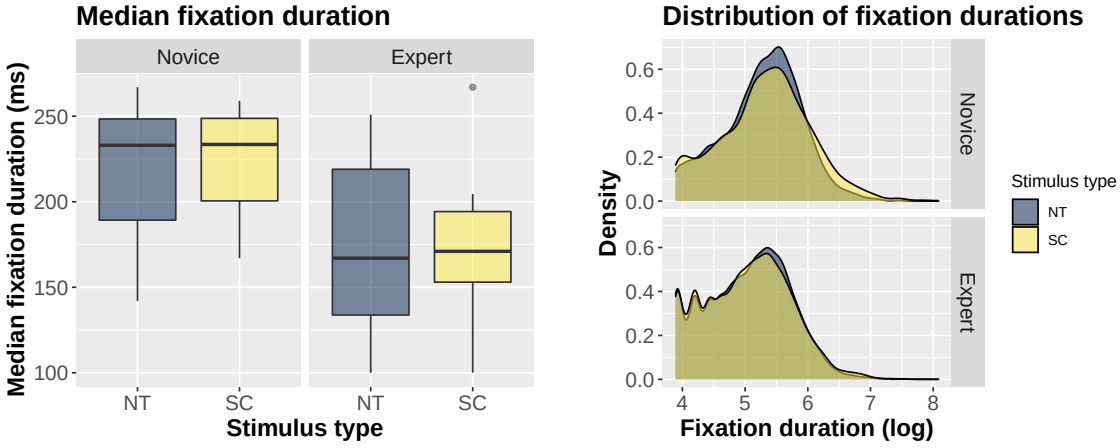
Figure 9.2: Fixation durations on NT stimuli per group of expertise

RQ1:
For comparing fixation durations between NT and SC, all fixations by a participant on the respective stimulus type were analyzed jointly. When examining novices, SC fixations were used from lesson 1, for experts from all stimuli. For novices, 6,010 NT fixations from 27 trials were compared to 2,374 SC fixations from 28 trials. The distribution of fixation durations is right-skewed, but median fixation durations follow a normal distribution. Dependent samples t-tests indicate no difference between the median fixation duration on both types of stimulus. A Kolmogorov-Smirnov test on the fixation duration distributions showed a significant difference between NT and SC fixations. For experts, 11,639 NT fixations from 45 trials were compared to 10,970 SC fixations from 38 trials. No differences were found between either the median fixation durations or the distributions of fixation duration (see table 9.3, figures 9.3 and 9.4).

RQ2:
For program L3_SC1/SC1, 2,825 fixations from novices were compared to 3,080 fixations from experts. For program L5_SC3/SC3, 3,271 novice fixations were tested against 4,073 expert fixations. The fixation duration distributions of the two groups were compared with Kolmogorov-Smirnov tests, which showed a significant difference for both programs (table 9.4 and figure 9.5a). Median fixation duration exhibits a normal distribution, so independent samples t-tests were used to compare novices and experts, and the two stimuli. The average median fixation duration of novices is higher than that of experts, however the difference is only significant for the second program (table 9.4 and figure 9.5b). The two stimuli are comparable in both regards, see table 9.5.

	Novices	**Experts**
Number of trials - NT	27	45
Number of trials - SC	28	38
Number of fixations - NT	6,010	11,639
Number of fixations - SC	2,374	10,970
p-value NT vs. SC (distribution of fixation durations)	**<0.001**	0.620
Mean median fixation duration - NT	219 ms (SD=41)	175 ms (SD=47)
Mean median fixation duration - SC	225 ms (SD=33)	171 ms (SD=41)
p-value NT vs. SC (median fixation duration)	0.625	0.625

Table 9.3: Fixation durations on NT and SC, p-values have been corrected for multiple testing



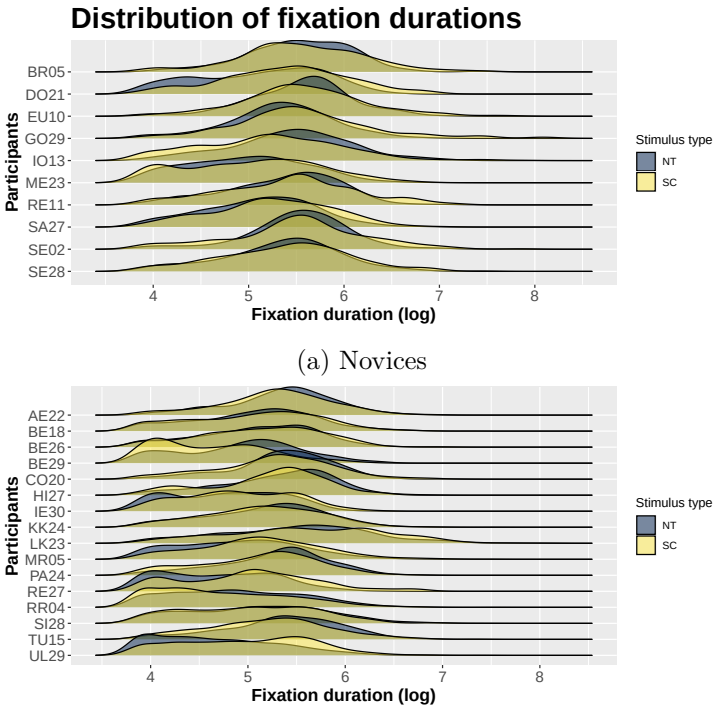
(a) Median fixation durations

(b) Distribution of fixation durations

Figure 9.3: Fixation durations on NT and SC per group of expertise

	L3_SC1/SC1	**L5_SC3/SC3**
Number of trials - novices	10	9
Number of trials - experts	13	13
Number of fixations - novices	2,825	3,271
Number of fixations - experts	3,080	4,073
p-value novices vs. experts (distribution of fixation durations)	**<0.001**	**<0.001**
Mean median fixation duration - stimulus	195 ms (SD=45)	187 ms (SD=45)
Mean median fixation duration - novices	213 ms (SD=45)	219 ms (SD=21)
Mean median fixation duration - experts	181 ms (SD=41)	166 ms (SD=44)
p-value novices vs. experts (median fixation duration)	0.140	**0.002**

Table 9.4: Fixation durations on the two programs viewed by both novices and experts, p-values have been corrected for multiple testing



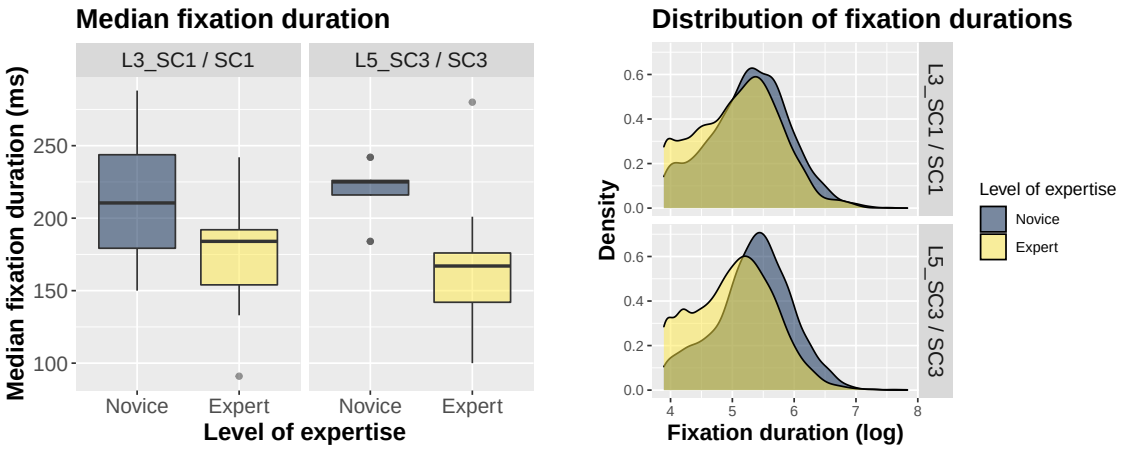
(a) Novices

(b) Experts

Figure 9.4: Distribution of fixation durations on NT and SC stimuli

Comparison	p-value
Distribution of fixation durations	0.427
Median fixation duration	0.578

Table 9.5: Results of the t-tests between the two programs viewed by both novices and experts



(a) Median fixation durations per SC

(b) Distribution of fixation durations per SC

Figure 9.5: Fixation durations on the two programs viewed by both novices and experts

Interpretation
Fixation durations vary substantially between readers and stimuli. Nevertheless the durations of 49 to 3,229 ms for reading the three English texts are well in accordance with results reported in literature and the distribution of fixation durations exhibits the characteristic right-skew. However, the average fixation duration of 183 ms [100..267] is lower than the 200-250 ms typically observed for reading English, see Holmqvist et al. [2011, 381,382], Rayner [1998, 373-376,392,393], and Rayner et al. [2005, 80-82] for reference values. Since none of the participants were English native speakers, the level of language proficiency probably has an additional impact on the fixation durations. Furthermore, fixation durations are highly dependent on the event detection approach, so the comparability to other studies is limited. Nevertheless, the three NT stimuli induced comparable distributions of fixation duration as well as average fixation durations. When comparing the fixation durations of novices and experts, the distribution of fixation duration on NT differs significantly and novices tend to have a higher average fixation duration. Fixation durations are highly idiosyncratic, so this was to be expected and has to be considered when interpreting the results for SC reading.

The distribution of fixation durations of novices is significantly different on NT than on SC, while the average durations are comparable on both types of stimuli. For experts, both distribution of and average fixation duration are comparable between stimuli. Thus for them the stimulus type did not strongly influence fixation durations.

The fixation duration distributions on the two SCs read by novices and experts were significantly different between the groups of expertise. The average fixation durations were comparable for the first stimulus, but on the second novices had a significantly higher average duration. Since these participants already had higher fixations durations on NT, this could be the result of the general reading behavior of these specific individuals and not an effect of expertise. However, the difference is huge and no difference was found for the first program, so it is very probable that the more difficult SC brought about a difference in fixation duration of novices and experts.

Overall, due to their idiosyncrasy, fixation durations are not a very indicative measure to study source code reading.

9.2.2 Number of fixations

For NT stimuli, 17,649 fixations were recorded in 72 trials from 26 participants, 10 novices and 16 experts. The number of fixations on these texts follow a normal distribution. Independent samples t-tests per text show no difference between novice and expert programmers see table 9.6 and figure 9.6a.

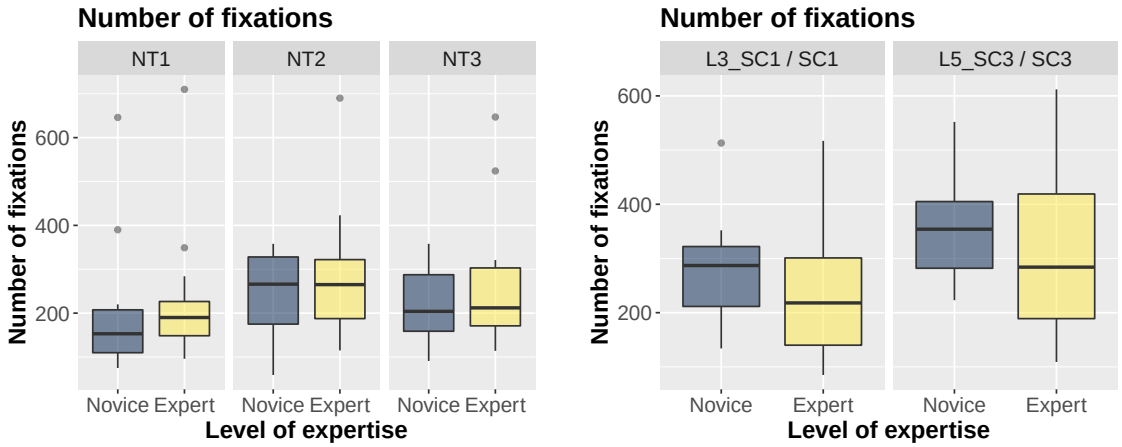
	NT1	**NT2**	**NT3**
Number of trials - novices	10	9	8
Number of trials - experts	15	15	15
Number of fixations - novices	2,109	2,159	1,742
Number of fixations - experts	3,357	4,269	4,013
Mean number of fixations - stimulus	219 (SD=159)	268 (SD=132)	250 (SD=130)
Mean number of fixations - novices	211 (SD=178)	240 (SD=110)	218 (SD=90)
Mean number of fixations - experts	224 (SD=151)	285 (SD=145)	268 (SD=147)
p-value novices vs. experts	0.853	0.404	0.328

Table 9.6: Number of fixations on NT stimuli

RQ2:
On the two programs read by both groups of expertise, 13,249 fixations were registered in 45 trials from 24 participants. Since the number of fixations per stimulus is normally distributed, independent samples t-tests were used for analysis. On both programs, the number of fixations is comparable between novices and experts, see table 9.7 and figure 9.6b.

	L3_SC1/SC1	L5_SC3/SC3
Number of trials - novices	10	9
Number of trials - experts	13	13
Number of fixations - novices	2,825	3,271
Number of fixations - experts	3,080	4,073
Mean number of fixations - stimulus	257 (SD=119)	334 (SD=140)
Mean number of fixations - novices	282 (SD=109)	363 (SD=102)
Mean number of fixations - experts	237 (SD=126)	313 (SD=162)
p-value novices vs. experts	0.364	0.384

Table 9.7: Number of fixations on the two programs viewed by both novices and experts



(a) NT stimuli

(b) SC stimuli viewed by novices and experts

Figure 9.6: Number of fixations

Interpretation
The number of fixations is strongly influenced by the stimulus and therefore only analyzed between participants on the same text. No differences were found between novices and experts, neither on the three NT stimuli nor on the two programs read by both groups. Thus with regard to the number of fixations needed to read these texts, novices and experts show a similar reading behavior.

9.2.3 Saccadic amplitude

Since saccadic amplitude depends on stimulus size and location of stimulus elements, analyses can only be carried out for RQ2, in which participants can be compared per stimulus.

RQ2:
For the two programs viewed by both novices and experts, 12,911 saccades were recorded from 10 novices and 14 experts. The distribution of saccadic amplitudes has a right-skew, median saccadic amplitude per trial and scanpath length follow a normal distribution. On the first program, saccadic amplitude ranges from 0.07 to 19.54° of visual angle, and has a median of 1.32° [0.76..2.56]. Total scanpath length lies between 203.1 and 1,219.3°, with a mean of 546.8° (SD=287.72). For L5_SC3/SC3, saccadic amplitude has a range of 0.05 to 23.76°, and a median of 1.29° [0.76..2.49]. Scanpath length spans from 230.4 to 1,625.5°, with a mean of 723.0° (SD=347.01). Using Kolmogorov-Smirnov tests, the distribution of saccadic amplitude was found to differ significantly between novices and experts on both programs. Independent samples t-tests did not show any differences between either median saccadic amplitude or scanpath length of novices and experts on any of the programs (table 9.8, figures 9.7 and 9.8).

	L3_SC1/SC1	**L5_SC3/SC3**
Number of trials - novices	10	9
Number of trials - experts	12	13
Number of saccades - novices	2,815	3,262
Number of saccades - experts	2,774	4,060
p-value novices vs. experts (distribution of saccadic amplitude)	**<0.001**	**<0.001**
Mean median amplitude - stimulus	1.35° (SD=0.26)	1.35° (SD=0.22)
Mean median amplitude - novices	1.26° (SD=0.14)	1.31° (SD=0.16)
Mean median amplitude - experts	1.41° (SD=0.25)	1.38° (SD=0.33)
p-value novices vs. experts (median amplitude)	0.200	0.662
Mean scanpath length - stimulus	546.84° (SD=287.72)	722.96° (SD=347.01)
Mean scanpath length - novices	587.44° (SD=262.21)	703.85° (SD=222.15)
Mean scanpath length - experts	513.00° (SD=314.71)	736.19° (SD=421.15)
p-value novices vs. experts (scanpath length)	0.662	0.818

Table 9.8: Saccadic amplitudes and scanpath lengths on the two programs viewed by both novices and experts, p-values have been corrected for multiple testing

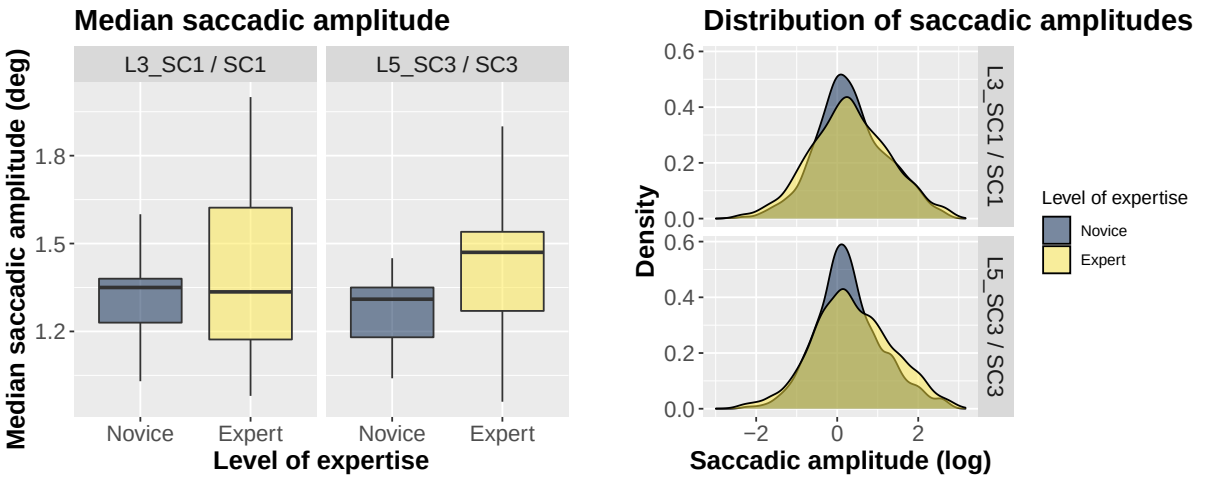
Interpretation
Like fixation duration and number of fixations, saccadic amplitude is a highly variable measure. The distribution of saccadic amplitudes during code reading shows the characteristic skew to the right [Holmqvist et al., 2011, 313], and differs significantly between novices and experts. The average extent of saccades and the scanpath lengths however are comparable. So while the novices and experts differ in how the saccadic amplitudes are distributed, average amplitude and scanpath length are not very distinctive with regard to programming expertise.

9.2.4 AOI coverage

Overall, data from 131 trials from 26 participants is available, 72 from novices and 59 from experts. On NT stimuli all lines were looked at and 98% of the elements. All hits on line-level were direct hits. On element-level 85% of the covered AOIs were hit directly by a fixation, the remaining AOIs were located close enough to a fixation to be perceived with the fovea. For SC, 86% of the lines were covered (94% of them directly) and 85% of the elements (67% directly). Only 3% of the total 28,704 fixations landed on empty space (see table 9.9).

	NT	**SC**
Overall	4%	3%
Novices	4%	2%
Experts	5%	4%

Table 9.9: Percentage of fixations on white space



(a) Median saccadic amplitudes per program (b) Distribution of saccadic amplitude per program

Figure 9.7: Saccadic amplitudes on the two programs viewed by both novices and experts

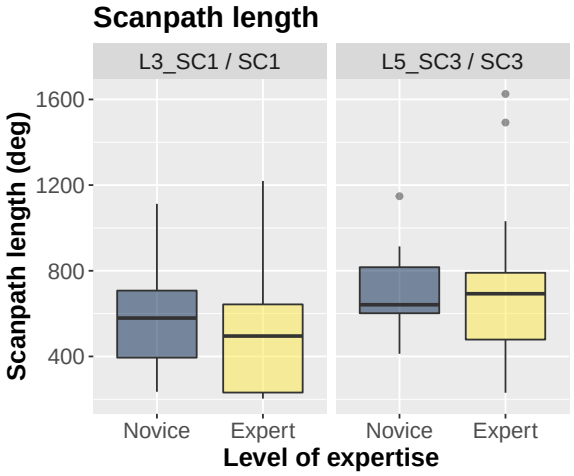


Figure 9.8: Scanpath lengths on the two programs viewed by both novices and experts

For analyzing AOI coverage on NT, 49 trials were used from 9 novices and 13 experts. In all three texts, all lines have been looked at by all participants, so no comparisons are carried out on line-level. AOI coverage on element-level is skewed to the left, over all NTs, it ranges from 82 to 100%, with a median of 99% [97..98]. A Kruskal-Wallis test showed a significant difference between the texts (p=0.001), so the texts are compared to each other with Mann-Whitney tests. Element coverage on NT2 is significantly different from NT1 and NT3, even though the difference of 1% can be regarded as negligible. NT1 and NT3 have comparable element coverage. No differences were found between the two groups of expertise (see tables 9.10 and 9.11, as well as figure 9.9).

	NT1	**NT2**	**NT3**
Number of trials - novices	9	8	8
Number of trials - experts	10	6	8
Median element coverage - stimulus	99% [97..99]	100% [100..100]	99% [99..100]
Median element coverage - novices	99% [97..99]	100% [100..100]	100% [99..100]
Median element coverage - experts	97% [97..99]	100% [100..100]	99% [96..99]
p-value novices vs. experts	0.522	0.522	0.333

Table 9.10: Element coverage on NT stimuli, p-values have been corrected for multiple testing

Comparison	NT1-NT2	NT1-NT3	NT2-NT3
p-value	**<0.001**	0.278	**0.018**

Table 9.11: Results of the Mann-Whitney tests to compare element coverage between NT stimuli, p-values have been corrected for multiple testing

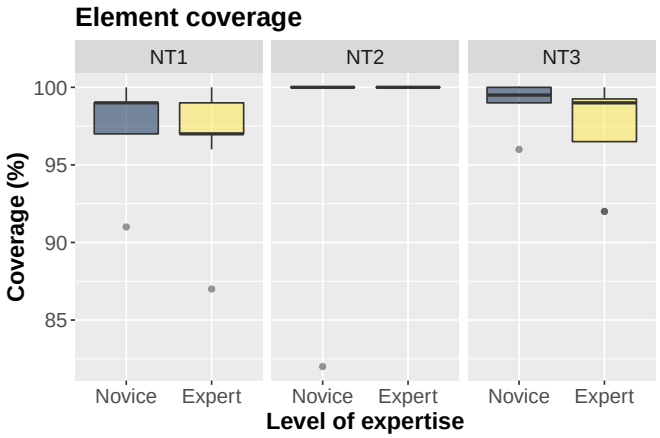
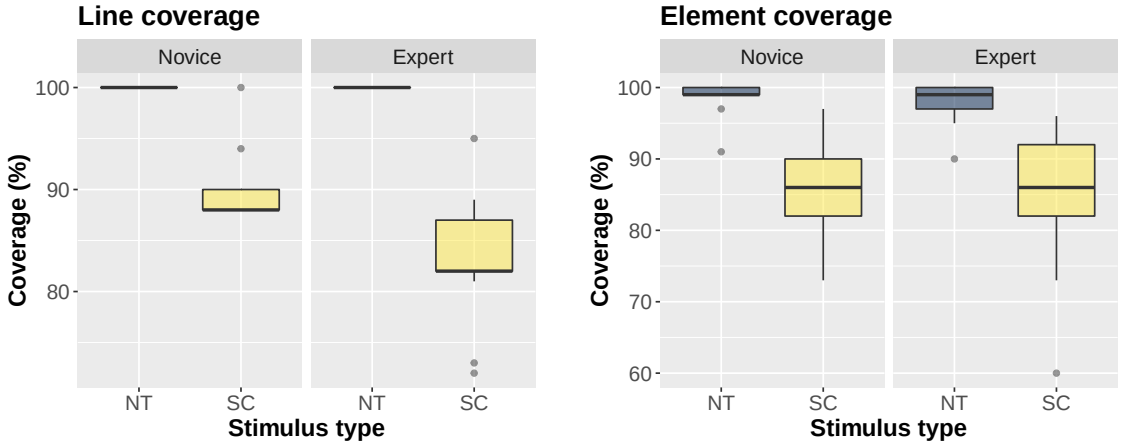


Figure 9.9: Element coverage per NT and expertise

RQ1:
In order to compare AOI coverage on NT and SC, 102 trials from 9 novices and 13 experts are used. AOI coverage on both AOI-levels has a leftward skew. Novices looked at all lines in NT stimuli, but on average only at 88% [88..90] of the lines in SC, the difference is significant. Similarly, they covered significantly more NT elements (median=99% [99..100]) than SC elements (median=86% [82..90]). The experts also looked at all NT lines, but on average only at 82% [82..87] of the SC lines. Furthermore, they covered more elements on NT (median=99% [97..100]) than on SC (median=86% [82..92]). The differences are significant for both AOI levels (see table 9.12 and figure 9.10).

	Novices	**Experts**
Number of trials - NT	25	24
Number of trials - SC	25	28
Median line coverage - NT	100% [100..100]	100% [100..100]
Median line coverage - SC	88% [88..90]	82% [82..87]
p-value NT vs. SC (line coverage)	**0.010**	**0.004**
Median element coverage - NT	99% [99..100]	99% [97..100]
Median element coverage - SC	86% [82..90]	86% [82..92]
p-value NT vs. SC (element coverage)	**0.010**	**0.004**

Table 9.12: AOI coverage on NT and SC stimuli, p-values have been corrected for multiple testing



(a) Line coverage per stimulus type and level of expertise

(b) Element coverage per stimulus type and level of expertise

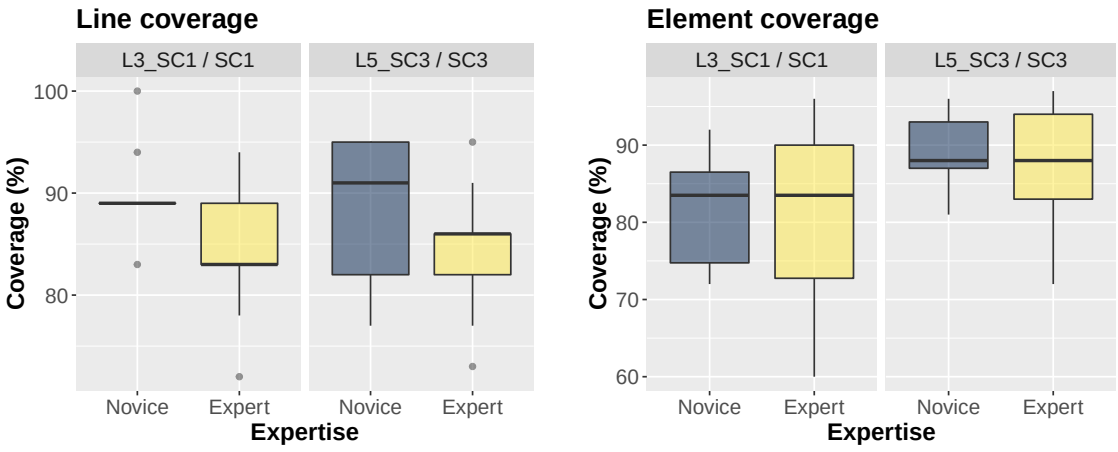
Figure 9.10: AOI coverage on NT and SC stimuli

RQ2:
When comparing AOI coverage of novices and experts on SC, 44 trials were analyzed from 10 novices and 14 experts. Both line and element coverage on SC are normally distributed, so independent samples t-tests were used for comparison. On the first program, novices covered significantly more lines than experts (novice mean=90%, SD=4, expert mean=84%, SD=6), but element coverage is comparable. For the second program, no differences were found. The two programs do not differ with regard to line coverage (p=0.861), but the second program has a significantly higher element coverage (mean = 88%, SD=7) than the first (mean = 81%, SD=9), p=0.035, see table 9.13 and figure 9.11. As before, p-values are corrected for multiple testing.

Additionally, it is analyzed which AOIs are not covered by gaze. While all NT lines were looked at, this was merely the case for 86% of the SC lines. Of the 173 SC lines that were skipped, 166 (96%) contained only a single closing bracket (table 9.14). Elements were skipped in both NT and SC stimuli (see tables 9.15 and 9.16).

	L3_SC1/SC1	**L5_SC3/SC3**
Number of trials - novices	10	9
Number of trials - experts	12	13
Mean line coverage - stimulus	87% (SD=6)	86% (SD=6)
Mean line coverage - novices	90% (SD=4)	88% (SD=7)
Mean line coverage - experts	84% (SD=6)	85% (SD=6)
p-value novices vs. experts (line coverage)	**0.035**	0.476
Mean element coverage - stimulus	81% (SD=9)	88% (SD=7)
Mean element coverage - novices	82% (SD=7)	90% (SD=5)
Mean element coverage - experts	81% (SD=11)	87% (SD=9)
p-value novices vs. experts (element coverage)	0.945	0.611

Table 9.13: AOI coverage on the two programs viewed by both novices and experts, p-values have been corrected for multiple testing



(a) Line coverage per stimulus

(b) Element coverage per stimulus

Figure 9.11: AOI coverage on the two programs viewed by both novices and experts

Stimulus	Line	Content	Number of skips	Number of occurrences	Percentage of skips
L1_SC1	5	`}`	7	9	78%
L1_SC1	6	`}`	9	9	100%
L3_SC1	6	`this.x2 = x2 ;`	1	10	10%
L3_SC1	8	`}`	4	10	40%
L3_SC1	17	`}`	5	10	50%
L3_SC1	18	`}`	8	10	80%
L5_SC3	9	`}`	3	9	33%
L5_SC3	15	`}`	2	9	22%
L5_SC3	17	`}`	5	9	56%
L5_SC3	21	`}`	5	9	56%
L5_SC3	22	`}`	8	9	89%
SC1	1	`public class Rectangle {`	2	12	17%
SC1	7	`this.y2 = y2 ;`	2	12	17%
SC1	8	`}`	9	12	75%
SC1	17	`}`	9	12	75%
SC1	18	`}`	12	12	100%
SC3	8	`this.currentSpeed = 0 ;`	1	13	8%
SC3	9	`}`	9	13	69%
SC3	15	`}`	3	13	23%
SC3	17	`}`	10	13	77%
SC3	21	`}`	7	13	54%
SC3	22	`}`	13	13	100%
SC5	1	`import java.util.ArrayList ;`	1	10	10%
SC5	11	`}`	2	10	20%
SC5	12	`}`	5	10	50%
SC5	13	`}`	3	10	30%
SC5	15	`}`	8	10	80%
SC5	21	`}`	10	10	100%
SC5	22	`}`	10	10	100%

Table 9.14: Lines which were skipped during SC reading, ordered according to stimulus

Element	Stimulus	Number of skips	Number of occurrences	Percentage of skips
with	NT1	6	38	16%
of	NT1, NT2	14	109	13%
do	NT3	2	16	12%
a	NT2, NT3	7	62	11%
in	NT1	2	19	11%
the	NT1, NT2, NT3	16	142	11%
it	NT1, NT3	3	35	9%
are	NT2	1	14	7%
at	NT2	1	14	7%
dung	NT2	1	14	7%
local	NT2	1	14	7%
obvious	NT2	1	14	7%
to	NT2, NT3	2	30	7%
years	NT2	1	14	7%
act	NT3	1	16	6%
and	NT1, NT3	4	70	6%
economy	NT3	1	16	6%
for	NT3	1	16	6%
is	NT3	1	16	6%
charcoal	NT1	1	19	5%
chinese	NT1	1	19	5%
first	NT1	1	19	5%
historians	NT1	1	19	5%
made	NT1	1	19	5%
most	NT1	1	19	5%
on	NT1	1	19	5%
or	NT1	2	38	5%

Table 9.15: Elements which were skipped during NT reading, ordered according to skipping proportion

Element	Stimulus	Number of skips	Number of occurrences	Percentage of skips
}	L1_SC1, L3_SC1, L5_SC3, SC1, SC3, SC5	217	362	60%
{	L1_SC1, L3_SC1, L5_SC3, SC1, SC3, SC5	94	226	42%
;	L1_SC1, L3_SC1, L5_SC3, SC1, SC3, SC5	216	571	38%
y2	L3_SC1, SC1	24	66	36%
)	L1_SC1, L3_SC1, L5_SC3, SC1, SC3, SC5	171	485	35%
>	SC5	6	20	30%
list2	SC5	3	10	30%
public	L1_SC1, L3_SC1, L5_SC3, SC1, SC3, SC5	57	201	28%
10	L3_SC1, L5_SC3, SC1	24	97	25%
if	L5_SC3, SC3, SC5	8	32	25%
import	SC5	5	20	25%
for	L1_SC2, SC5	7	29	24%
this.y2	L3_SC1, SC1	5	22	23%
type	SC3	3	13	23%
x2	L3_SC1, SC1	13	56	23%
2	L1_SC1	2	9	22%
200	L5_SC3	2	9	22%
(	L1_SC1, L3_SC1, L5_SC3, SC1, SC3, SC5	76	367	21%
0	L3_SC1, L5_SC3, SC1, SC3	9	44	20%
1	L1_SC3	2	10	20%
tp	L5_SC3, SC3	7	35	20%
y	SC5	2	10	20%
args	L1_SC1, L3_SC1, SC3	6	32	19%
*	L3_SC1, SC1	4	22	18%
+	L5_SC3, SC3	4	22	18%
=	L3_SC1, SC1, SC3, SC5	25	140	18%
class	SC1, SC5	4	22	18%
int	L3_SC1, L5_SC3, SC1, SC3, SC5	26	146	18%
kmh	L5_SC3, SC3	4	22	18%
this.x2	L3_SC1, SC1	4	22	18%
,	L3_SC1, L5_SC3, SC1, SC3, SC5	53	313	17%
]	L1_SC1, L3_SC1, SC1, SC3, SC5	14	84	17%
<	SC5	5	30	17%
5	SC1	4	24	17%
[	SC1, SC3, SC5	9	55	16%
i	SC5	3	20	15%
List	SC5	3	20	15%
this.y1	L3_SC1, SC1	3	22	14%
x1	L3_SC1, SC1	3	22	14%
y1	L3_SC1, SC1	9	66	14%
Rectangle	SC1	3	24	12%
2.0	L1_SC2	1	9	11%
even	L1_SC2	1	9	11%
producer	L5_SC3	1	9	11%

Table 9.16 continued from previous page

Element	Stimulus	Number of skips	Number of occurrences	Percentage of skips
`1`	L1_SC3	1	10	10%
`9.1`	SC5	1	10	10%
`double`	SC5	2	20	10%
`java.util.Arraylist`	SC5	1	10	10%
`list1`	SC5	1	10	10%
`this.height`	L3_SC1	1	10	10%
`-`	L1_SC3, SC1	2	22	9%
`else`	L5_SC3, SC3	2	22	9%
`System.out.println`	L3_SC1, SC1	2	22	9%
`p`	SC3	2	26	8%
`rect2`	SC1	1	12	8%
`return`	SC1	1	12	8%
`this.currentSpeed`	SC3	1	13	8%
`this.x1`	SC1	1	12	8%
`Vehicle`	SC3	1	13	8%

Table 9.16: Elements which were skipped during SC reading, ordered according to skipping proportion

Interpretation

For NT reading it was found that significantly more elements were covered on NT2 than on the other two texts, but since the difference in average coverage only amounts to 1%, this finding is rather inconsequential. Novices and experts have comparable element coverage on all three texts. When reading NT, all lines were looked at, while 14% of the lines in SC were skipped. Since many participants answered the comprehension question correctly despite skipping at least one line, it is possible to fully understand a program without fixating all lines. The lines which were skipped contained almost exclusively just a single closing bracket. Such lines form somewhat smaller AOIs, so they might be perceived well enough in the peripheral view to be deemed unnecessary to be looked at directly or participants just expect them to be there, since the instruction slide informed them that all programs compile without errors. In lesson 1, SC2 and SC3 are pseudocodes and do not contain closing brackets. No lines were skipped in these programs. Program L1_SC1 however ended with two lines each consisting of only a closing bracket. The first one was skipped by seven of nine novices, the second one by all nine. In the two advanced novice programs, with one exception, novices skipped only lines with a closing bracket. Thrice, experts skipped the first line of a program and in three other cases, they did not look at the last of the four structurally similar assignments in the constructor. Other than that, they only left out lines with closing brackets. In all programs which included closing brackets, the last line was ignored most. Experts did not even once look at a final bracket in any program. These findings affirm the concept of treating lines with only a single bracket as skippable AOIs when studying how the gaze aligns with the Execution Order model.

Both novices and experts skip more elements on SC than on NT. Words that are repeatedly skipped in NT are as expected mostly frequent and short words like “of”, “a”, and “the”. In SC, many of the elements that were not looked at consist of only one character. The elements that were skipped the most are the separators “}”, “{”, and “;”. This is in line with the findings by Busjahn et al. [2014a] that separators receive substantially less visual attention than the other types of lexical elements and Blascheck & Sharif [2019, 4] that closing brackets are hardly focused on. The keyword that was skipped most often is `public`, closely followed by `if`. One reason for the lower element coverage in SC is probably the huge amount of short elements. Median element length in NT stimuli is 4 [3..8] characters, in lesson 1 it is 2 [1..5] characters, and for expert SC merely 1 [1..5] character. Median length of skipped words in NT is 3 [2..3] characters, in lesson 1 and expert SC 1 [1..1] character. Several short elements can be perceived with a single fixation, so less elements have to be fixated. Furthermore, the employed programming language prescribes a certain structure for the program, so many elements can be presumed to be present and probably do not have to be perceived foveally. This accounts e.g. for the many skipped `public`-elements.

By and large, novice and expert programmers do not differ much in what proportion of SC they cover with gaze.

AOI coverage is a measure that is strongly influenced by the type of text. Programming expertise however is not a crucial factor for how much of a SC is covered.

9.2.5 First visit to `main`

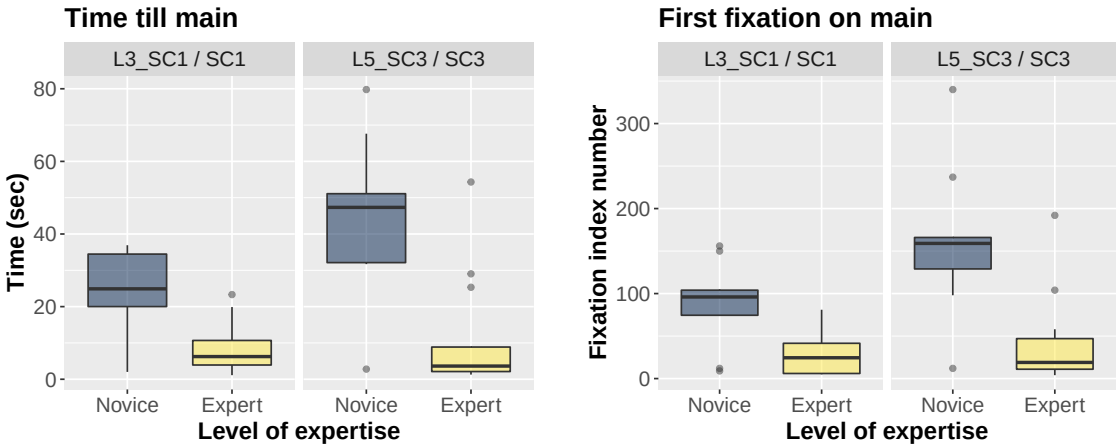
RQ2:
For program L3_SC1/SC1 10 trials were analyzed from novices and 12 from experts, for program L5_SC3/SC3 9 trials from novices and 13 from experts. Neither time until the gaze landed in the `main`-method for the first time nor the number of the first fixation on `main` are normally distributed, so Mann-Whitney tests are used. When reading the first program, novices on average visited the `main`-method after 25 sec [20..34] and with the 96th [74..104] fixation, experts already after 6 sec [4..11] and with the 24th [6..42] fixation. In both cases the difference between novices and experts is significant. Similarly, on the second program, novice gaze arrived in `main` on average after 47 sec [32..51] and with the 159th [129..166] fixation, experts after 4 sec [2..9] and with the 19th [11..47] fixation. Again, the differences are significant (see table 9.17 and figure 9.12). For program L3_SC1/SC1, novices spent on average the first 35% [32..46] of the total trial duration somewhere else, before looking at `main`, for program L5_SC3/SC3 even 52% [46..59]. Experts however, spent only the first 14% [6..26] and 7% [4..12] of the total time elsewhere, before attending to `main`.

	L3_SC1/SC1	**L5_SC3/SC3**
Number of trials - novices	10	9
Number of trials - experts	12	13
Median time till main - stimulus	11 sec [5..24]	17 sec [3..46]
Median time till main - novices	25 sec [20..34]	47 sec [32..51]
Median time till main - experts	6 sec [4..11]	4 sec [2..9]
p-value novices vs. experts (time)	**0.007**	**0.003**
Median number of 1st fixation on main - stimulus	44 [10..88]	52 [15..156]
Median number of 1st fixation on main - novices	96 [74..104]	159 [129..166]
Median number of 1st fixation on main - experts	24 [6..42]	19 [11..47]
p-value novices vs. experts (number of 1st fixation)	**0.004**	**0.005**

Table 9.17: First visit to `main` for the two programs viewed by both novices and experts

Neither time nor number of fixations of the first dwell on `main` show a normal distribution. For the first program, novices have a median dwell time of 1 sec [0..9], experts of 4 sec [2..5]. On the second program, the median dwell times are 4 sec [1..5] for novices and almost identical 4 sec [3..7] for experts. The first dwell on program L3_SC1/SC1 on average included 6 [2..42] fixations for novices and 16 [8..21] for experts. For program L5_SC3/SC3, the first dwell consisted on average of 15 [5..23] fixations for novices and 19 [14..31] for experts. Independent samples t-tests showed no significant differences between novices and experts regarding the duration of the first dwell on `main` or the number of fixations in it (see table 9.18 and figure 9.13).

Interpretation
As claimed during the interviews, experts look at the `main`-method soon after they started reading the program. Novices on the other hand took on average at least four times as much time to reach the `main`. The `main`-method in Java is usually located at the bottom of the class. Experts visiting it so early after stimulus onset suggests that they adapted their reading strategy to this type of text. This ties in with experts only looking on average at 36% [27..64] of the lines above `main` before arriving there, novices at 82% [82..90]. After the gaze lands on the `main` however, novices and experts do not differ in the duration of their first dwell there. The measures time and number of fixations until the `main`-method is visited prove to be very meaningful for marking differences in the code reading behavior of novices and experts.



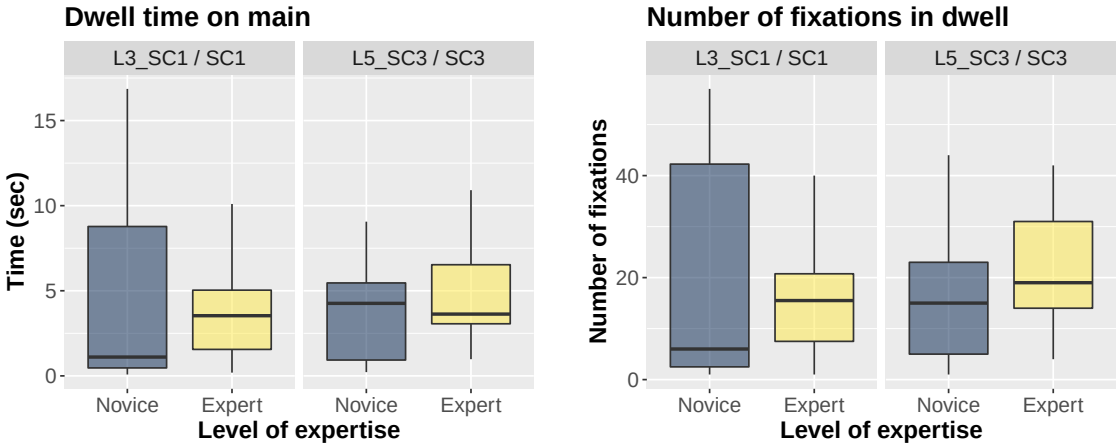
(a) Time until the first fixation hits `main` in sec

(b) Index number of first fixation on `main`

Figure 9.12: First visit to the `main`-method for the two programs viewed by both novices and experts

	L3_SC1/SC1	**L5_SC3/SC3**
Number of trials - novices	10	9
Number of trials - experts	12	13
Median dwell time - stimulus	3 sec [0..5]	4 [2..6]
Median dwell time - novices	1 sec [0..9]	4 sec [1..5]
Median dwell time - experts	4 sec [2..5]	4 sec [3..7]
p-value novices vs. experts (time)	0.539	0.556
Median number of fixations on main - stimulus	10 [3..24]	18 [10..27]
Median number of fixations on main - novices	6 [2..42]	15 [5..23]
Median number of fixations on main - experts	16 [8..21]	19 [14..31]
p-value novices vs. experts (fixations)	0.667	0.367

Table 9.18: First dwell on the `main`-method for the two programs viewed by both novices and experts



(a) Dwell time on `main` in sec

(b) Number of fixations on `main`

Figure 9.13: First dwell on the `main`-method for the two programs viewed by both novices and experts

9.3 Event-sequence-based measures

9.3.1 Reading direction

In total, the full AOI sequences contain 27,569 items on line-level and 27,426 on element-level. For both AOI-levels only 3% of the items in the AOI sequences have a stopover on white space, thus the vast majority of the movements from one item to the next in the AOI sequences are actual saccades. Looking at the cases with intermediate fixations, the median amount of fixations in-between AOIs is 1 [1..1]. Less than 1% of the movements were interrupted by more than one fixation.

There are 49 NT trials from 9 novices and 13 experts. Overall, 8% of the movements between lines were directed forward, 6% backward, and 86% remained on the same line. On element-level, 56% were directed forward, 20% backward, 24% were stationary. The proportion of linear movements is normally distributed on both line- and element-level. A two-way ANOVA showed no differences between stimuli (p=0.354), nor between levels of expertise (p=0.076) for line-AOIs. On element-level, the stimuli are also comparable (p=0.093), but novices read the texts significantly more linearly than experts (p=0.012). See table 9.19 and figure 9.14 for details.

	NT1	**NT2**	**NT3**
Number of trials - novices	9	8	8
Number of trials - experts	10	6	8
Mean linear proportion (line-level) - stimulus	95% (SD=4)	93% (SD=4)	94% (SD=4)
Mean linear proportion (line-level) - novices	97% (SD=4)	94% (SD=3)	94% (SD=4)
Mean linear proportion (line-level) - experts	93% (SD=3)	92% (SD=4)	94% (SD=3)
Mean linear proportion (element-level) - stimulus	83% (SD=7)	79% (SD=7)	82% (SD=7)
Mean linear proportion (element-level) - novices	82% (SD=5)	83% (SD=4)	87% (SD=5)
Mean linear proportion (element-level) - experts	74% (SD=6)	82% (SD=9)	80% (SD=6)

Table 9.19: Proportion of linear reading on NT stimuli

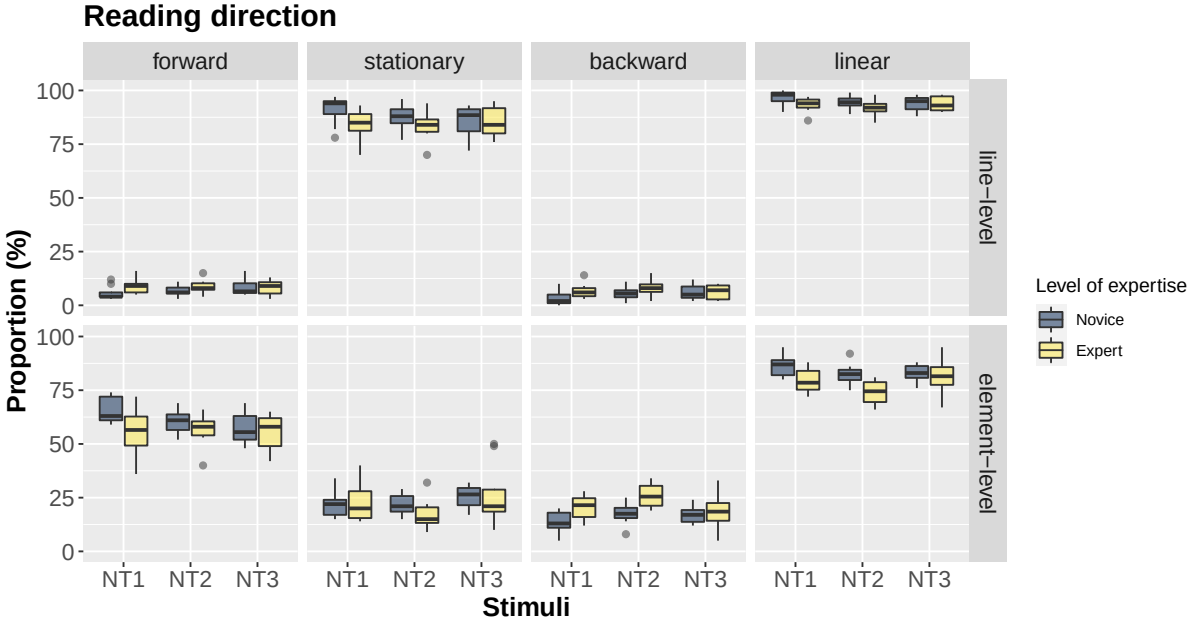


Figure 9.14: Proportion of reading directions on NT stimuli per group of expertise

RQ1:
Data from 102 trials, 9 novices and 13 experts, was analyzed for RQ1. The proportion of linear reading on lines as well as on elements is normally distributed, so dependent samples t-tests were used to compare NT and SC. On both AOI levels, novices and experts exhibit a significantly higher proportion of linear reading on NT than on SC (see table 9.20 and figure 9.15).

	Novices	**Experts**
Number of trials - NT	25	24
Number of trials - SC	25	28
Mean linear proportion (line-level) - NT	95% (SD=3)	93% (SD=4)
Mean linear proportion (line-level) - SC	84% (SD=3)	79% (SD=3)
p-value NT vs. SC	**<0.001**	**<0.001**
Mean linear proportion (element-level) - NT	84% (SD=5)	79% (SD=7)
Mean linear proportion (element-level) - SC	65% (SD=4)	73% (SD=3)
p-value NT vs. SC (linear proportion element-level)	**<0.001**	**<0.001**

Table 9.20: Proportion of linear reading on NT and SC stimuli, p-values have been corrected for multiple testing

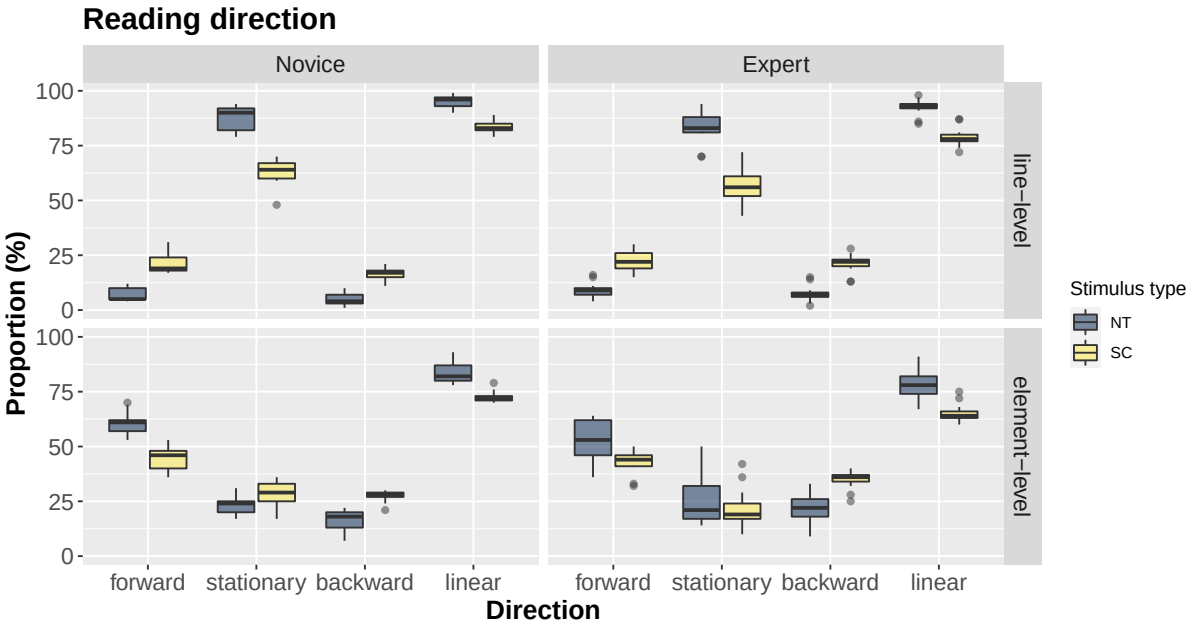


Figure 9.15: Proportion of reading directions on NT and SC stimuli per group of expertise

RQ2:
For analyzing the reading directions on SC, 44 trials from 10 novices and 14 experts were used. The proportion of linear reading exhibits a normal distribution, both on line- and element-level. On the first SC, independent samples t-tests show that the proportion of linear reading of novices and experts is comparable on lines and elements. However, before correcting for multiple testing, the higher proportion of linear reading on line-level found in novices was significantly so (p-value was 0.045). On the second program, novices read significantly more linearly than experts, both with regard to lines and elements (see table 9.21 and figure 9.16). The proportion of linear line reading is higher for the first program, but the difference only borders significance (p=0.051, before correcting for multiple testing p=0.029). On element-level, both programs are comparable (p=0.153).

	L3_SC1/SC1	L5_SC3/SC3
Number of trials - novices	10	9
Number of trials - experts	12	13
Mean linear proportion (line-level) - stimulus	86% (SD=5)	80% (SD=4)
Mean linear proportion (line-level) - novices	86% (SD=4)	83% (SD=3)
Mean linear proportion (line-level) - experts	82% (SD=4)	79% (SD=5)
p-value novices vs. experts (linear proportion line-level)	0.063	**0.028**
Mean linear proportion (element-level) - stimulus	70% (SD=7)	69% (SD=4)
Mean linear proportion (element-level) - novices	69% (SD=4)	72% (SD=4)
Mean linear proportion (element-level) - experts	65% (SD=5)	67% (SD=5)
p-value novices vs. experts (linear proportion element-level)	0.093	**0.040**

Table 9.21: Proportion of linear reading on the two programs viewed by both novices and experts, p-values have been corrected for multiple testing

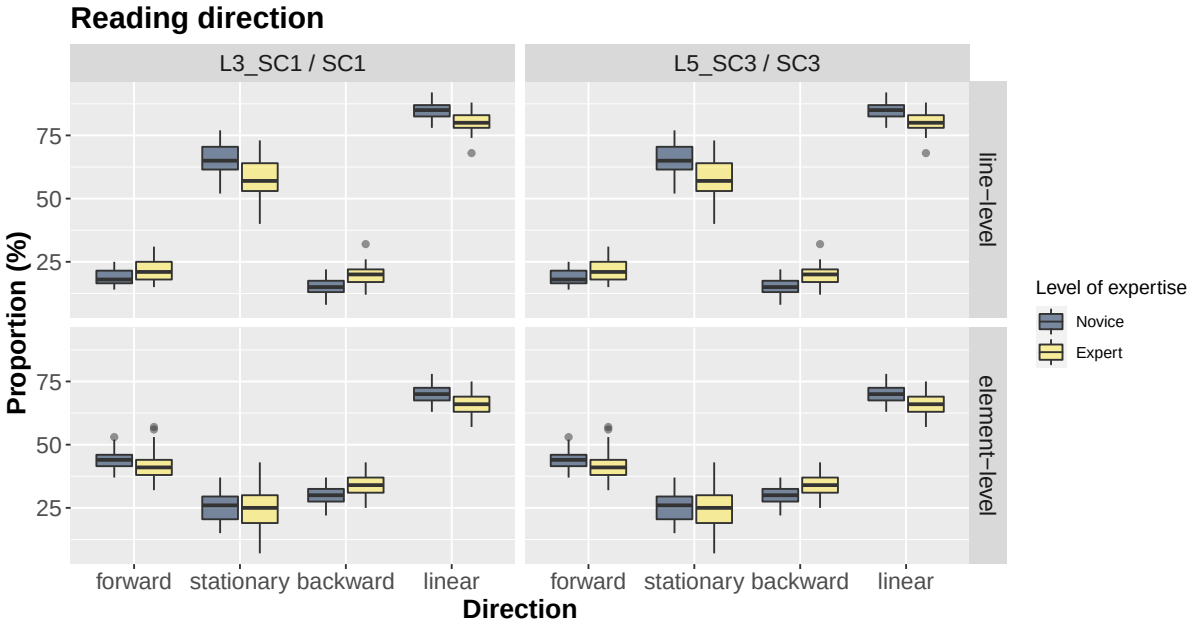


Figure 9.16: Proportion of reading directions on the two programs viewed by both novices and experts

Interpretation

When reading NT, as expected, the gaze mostly stays on a line and occasionally moves forward to the next one. On element-level, the gaze moves forward with about half of the movements, about a quarter remains on the same word. 20% of the movements on element-level were directed backwards. The regression rate when reading English usually ranges around 10 to 15% [Rayner, 1998, 375], [Rayner et al., 2005, 80]. Considering that none of the participants were native English speakers and the texts were probably read very thoroughly in order to answer the comprehension questions, the slightly higher rate found in the experiment can be deemed normal. Besides, the movements in AOI sequences are not completely equal to saccades, so the rates are only roughly comparable. No difference in proportion of linear reading was found between the English texts. On line-level novices and experts were also comparable, but on element-level the novices showed a significantly higher proportion of linear NT reading than experts.

The NT stimuli were read considerably more linearly than the SCs, both by novices and experts and on both AOI levels. This aligns well with the results by Crosby & Stelovsky [1990] and Busjahn

et al. [2011] that SC induces considerably more backward movements than NT. While the English texts are inherently linear, the SCs include constructs like loops and calls to previously defined methods, thus the order in which the text is presented does not always correspond to its execution. If a participant wants to trace the control flow, an integral information about the program, non-linear movements are needed to facilitate understanding. Furthermore, the SC was probably at least partly more difficult to comprehend than the NT, so the regressive movements can also be caused by difficulties in understanding the programs. Some participants might also exhibit a certain amount of thrashing, i.e. the gaze partly moves around seemingly at random [Bednarik et al., 2014, 36,37], [Simon, 2014, 28]. Such thrashing possibly occurs when someone arbitrarily looks around for any clue as to what the program does and can also result in more backward movements, even if the general reading approach is rather linear.

When reading SC, novices consistently read more linearly than experts, but only the differences for the second program are significant. The novice programmers already had a tendency to read the NT stimuli on element-level more linearly than experts, so besides the lesser programming proficiency their general reading approach might to a certain degree contribute to this finding.

Moving linearly from one AOI to another was found to be very characteristic of NT reading, but not of code reading. Thus reading direction is an adequate measure to capture the difference of reading these types of stimuli. As for programming expertise, the proportion of linear reading on SC is higher for novices than for experts, but the difference between the two levels of expertise is not as pronounced as between the stimuli types.

9.3.2 Model occurrence and model similarity

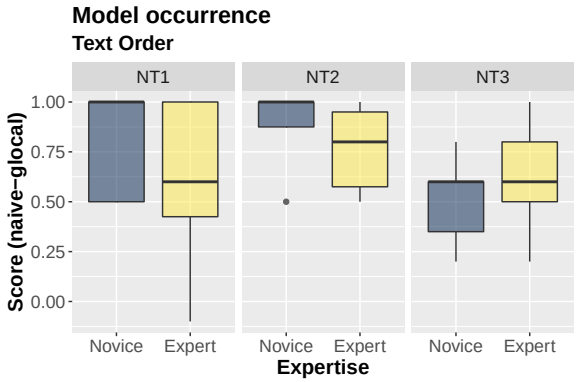
The lengths of full AOI sequences range from 25 to 640 items, with a median length of 193 [108..295] items. Collapsed sequences on the other hand have lengths of 4 to 285 items, with a median length of 44 [24..64] items.

For NT, gaze sequences from 49 trials, recorded from 9 novices and 13 experts, are compared to Text Order sequences. Naive glocal scores are not normally distributed. A Kruskal-Wallis test showed no difference between the naive glocal scores of the three stimuli ($p=0.061$), a Mann-Whitney test no difference between the scores of novices and experts ($p=0.476$). Both naive and dynamic global scores on NT follow a normal distribution. Two-way ANOVAs showed that naive and dynamic global scores are comparable between stimuli (naive $p=0.222$, dynamic $p=0.181$) and naive global scores also between novices and experts ($p=0.250$). Dynamic global scores differ significantly between novices and experts ($p=0.040$). Independent samples t-tests show that on NT1 novices have a significantly higher dynamic global Text Order score than experts ($p=0.028$), but no difference was found for NT2 ($p=0.429$) and NT3 ($p=0.461$). The p-values for comparing dynamic global scores between the two levels of expertise are corrected for multiple testing. The number of model repetitions to achieve the best dynamic alignment of gaze and model is not normally distributed and ranges from 1 to 22, with a median of 6 [2..10] repetitions, see table 9.22 and figure 9.17.

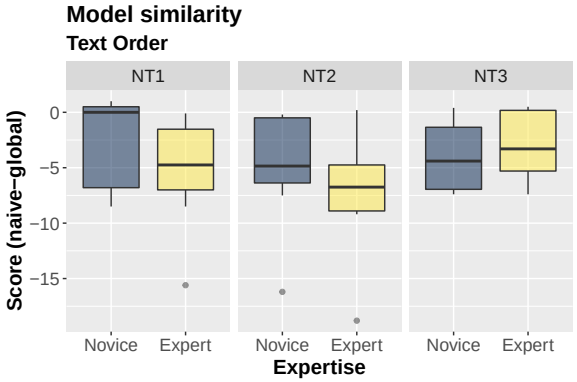
RQ1:
For comparing average Text Order scores on NT and SC stimuli, data from 49 trials, 9 novices and 13 experts was used. Median scores are not normally distributed, so Wilcoxon matched-pairs tests were used to compare scores between NT and SC stimuli. For novices no significant differences were found between the Text Order scores on the two types of stimuli. For experts however, both naive glocal and dynamic global Text Order scores are significantly higher for NT than for SC (see table 9.23 and figure 9.18). Figure 9.19 illustrates that while for novices it is rather arbitrary whether the Text Order model sequence has a higher score in NT or SC, for experts the scores are in general much higher for NT than for SC.

		NT1	NT2	NT3
Number of trials - novices		9	8	8
Number of trials - experts		10	6	8
Median score (naive) - stimulus	glocal	0.7 [0.5..1]	1 [0.6..1]	0.6 [0.4..0.6]
Median score (naive) - novices	glocal	1 [0.5..1]	1 [0.9..1]	0.6 [0.4..0.6]
Median score (naive) - experts	glocal	0.6 [0.4..1]	0.8 [0.6..1]	0.6 [0.5..0.8]
Mean score (naive) - stimulus	global	-3.8 (SD=4.5)	-6.2 (SD=5.7)	-3.6 (SD=3)
Mean score (naive) - novices	global	-2.3 (SD=4)	-5.1 (SD=5.3)	-4.1 (SD=3.1)
Mean score (naive) - experts	global	-5.2 (SD=4.7)	-7.6 (SD=6.4)	-3 (SD=3.1)
Mean score (dynamic) - stimulus	global	0.3 (SD=4.5)	0.3 (SD=5.7)	0.1 (SD=3)
Mean score (dynamic) - novices	global	0.5 (SD=4)	0.3 (SD=5.3)	0.1 (SD=3.1)
Mean score (dynamic) - experts	global	0.1 (SD=4.7)	0.2 (SD=6.4)	0.2 (SD=3.1)
Median number of repetitions - stimulus		5 [2..10]	8 [4..12]	6 [4..9]
Median number of repetitions - novices		2 [2..9]	7 [3..9]	7 [5..10]
Median number of repetitions - experts		6 [4..11]	10 [7..14]	6 [1..8]

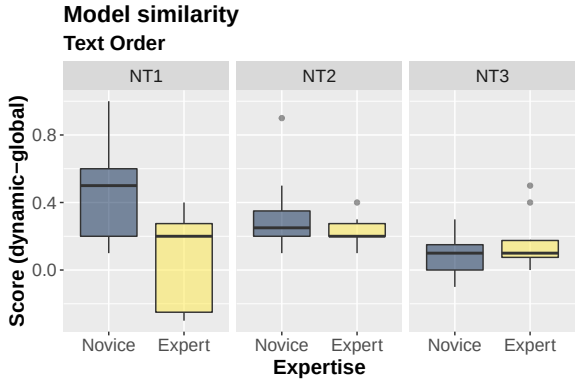
Table 9.22: Occurrence of and similarity to Text Order sequences on NT stimuli



(a) Naive glocal scores



(b) Naive global scores



(c) Dynamic global scores

Figure 9.17: Text Order scores per NT stimulus

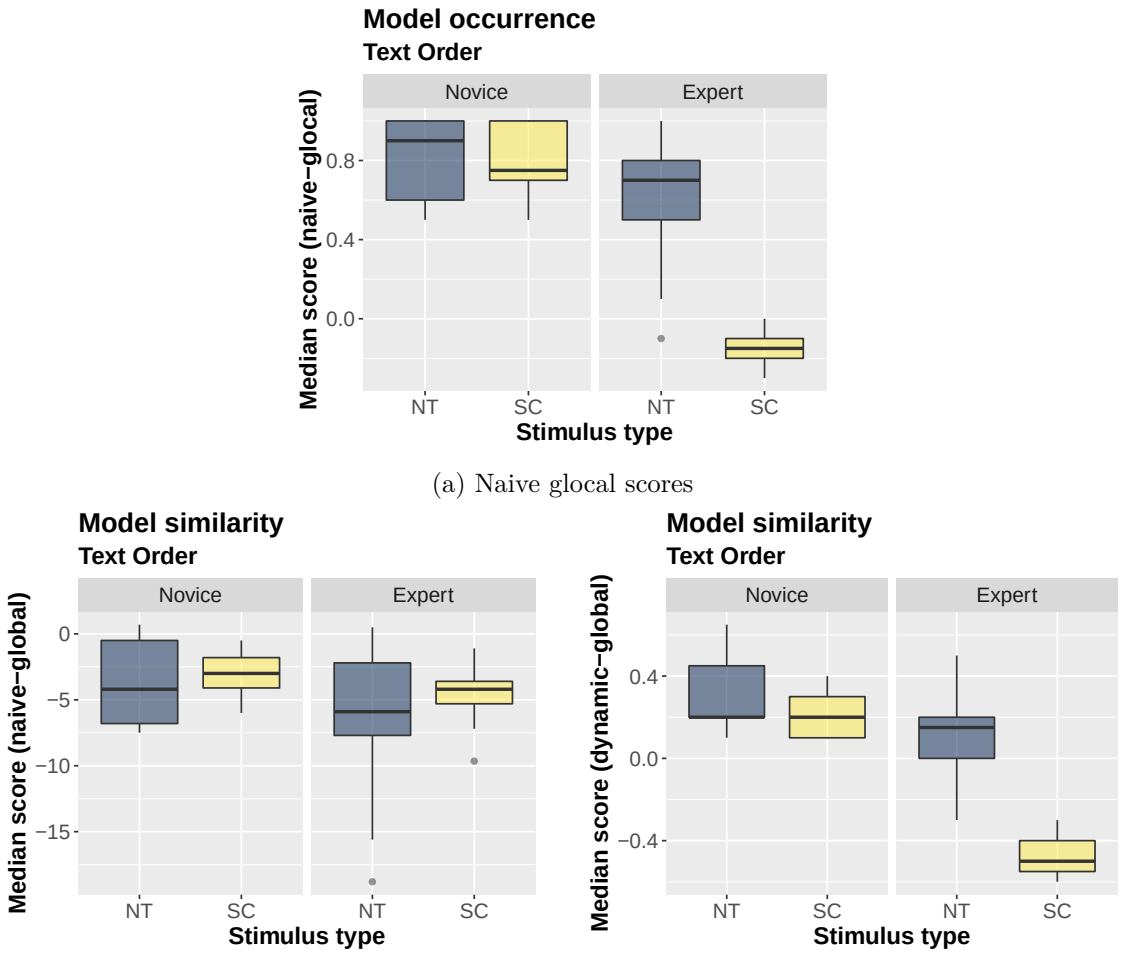
		Novices	Experts
Number of trials - NT		25	24
Number of trials - SC		25	28
Median score (naive) - NT	glocal	0.9 [0.6..1]	0.7 [0.5..0.8]
Median score (naive) - SC	glocal	0.8 [0.7..1]	-0.2 [-0.2..-0.1]
p-value NT vs. SC	glocal	0.944	**0.003**
Median score (naive) - NT	global	-4.2 [-6.8..-0.5]	-5.9 [-7.7..-2.2]
Median score (naive) - SC	global	-3 [-4.1..-1.8]	-4.2 [-5.3..-3.6]
p-value NT vs. SC	global	0.859	0.542
Median score (dynamic) - NT	global	0.2 [0.2..0.4]	0.2 [0..0.2]
Median score (dynamic) - SC	global	0.2 [0.1..0.3]	-0.5 [-0.6..-0.4]
p-value NT vs. SC	global	0.106	**0.002**
Median number of repetitions - NT	global	6 [2.5..9]	6 [5..11]
Median number of repetitions - SC	global	5 [4..7]	5 [4..6]

Table 9.23: Occurrence of and similarity to Text Order sequences on NT and SC stimuli

For comparing Text and Execution Order scores on SC, 63 trials from 10 novice and 16 expert programmers were analyzed. Since no comparison to NT is needed, 10 additional trials were included, which belong to participants without NT recordings of sufficient data quality. None of the scores exhibit a normal distribution, so Wilcoxon matched-pairs tests are used to test for differences. For novices, only the naive global scores for Text Order are significantly lower than Execution Order scores. Glocal and dynamic global scores are comparable between the two models. For experts, naive glocal scores for the Text Order model are significantly higher than for the Execution Order model and naive global scores for the Text Order model are significantly lower than for the Execution Order model. The difference between dynamic global scores is not significant, but only just (see table 9.24 and figure 9.20).

		Novices	Experts
Number of trials - SC		28	35
Median score (naive) - Text Order	glocal	0.7 [0.5..1]	-0.2 [-0.2..0]
Median score (naive) - Execution Order	glocal	0.6 [0.4..1]	-0.2 [-0.3..-0.1]
p-value Text vs. Execution Order	glocal	0.631	**0.031**
Median score (naive) - Text Order	global	-3 [-5.8..-1.7]	-3.9 [-5.4..-2.1]
Median score (naive) - Execution Order	global	-2.2 [-3.8..-1]	-1.8 [-4.5..-1.2]
p-value Text vs. Execution Order	global	**0.036**	**0.029**
Median score (dynamic) - Text Order	global	0.2 [0.1..0.3]	-0.5 [-0.6..-0.4]
Median score (dynamic) - Execution Order	global	0.3 [0.1..0.3]	-0.6 [-0.7..-0.5]
p-value Text vs. Execution Order	global	0.297	0.055
Median number of repetitions - Text Order	global	6 [2..9]	6 [5..11]
Median number of repetitions - Execution Order	global	5 [4..7]	5 [4..6]

Table 9.24: Occurrence of and similarity to Text and Execution Order sequences on SC stimuli



(a) Naive glocal scores

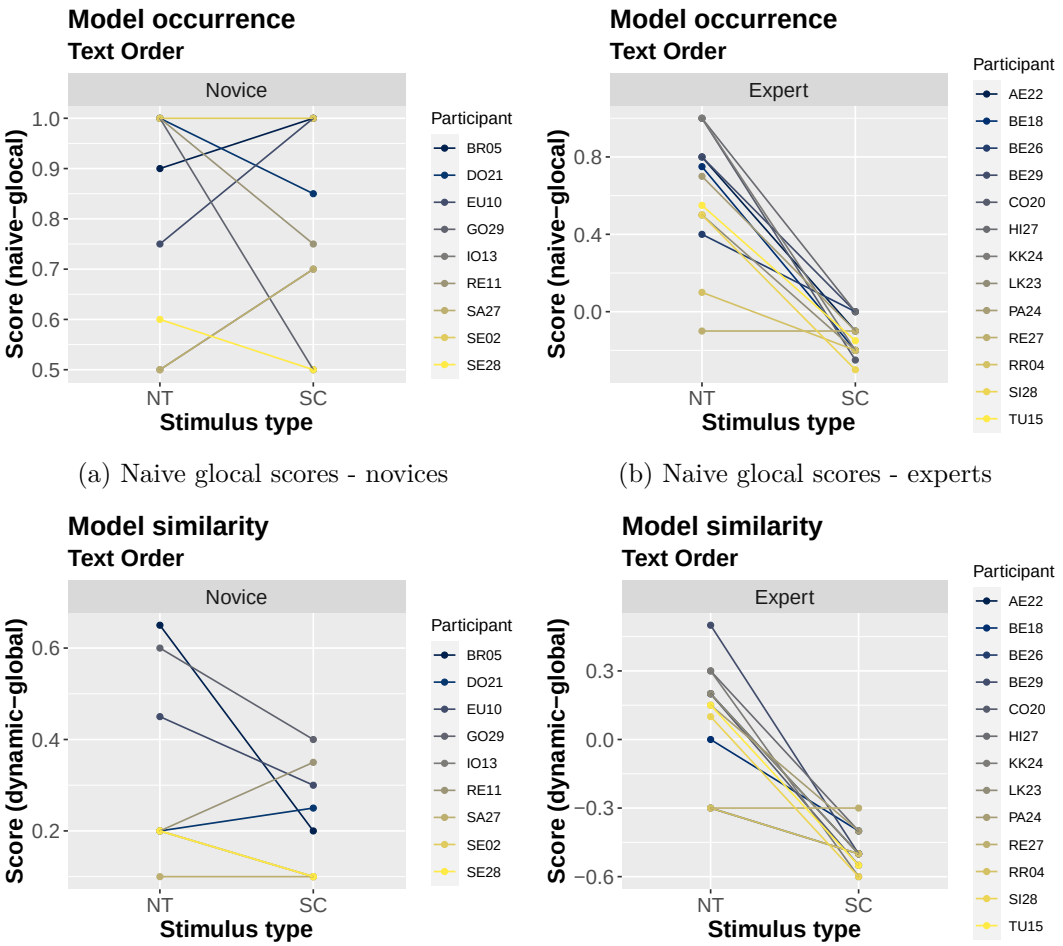
(b) Naive global scores

(c) Dynamic global scores

Figure 9.18: Text Order scores on NT and SC stimuli

RQ2:
44 trials from 10 novices and 14 experts were analyzed. Neither the different scores nor the number of model repetitions are normally distributed, so Mann-Whitney tests were used to compare the results between novices and experts. With regard to glocal alignments, no significant differences were found between the two groups of expertise on any of the two programs. For global scores, solely the dynamic Execution Order score on program L3_SC1/SC1 was significantly higher for experts than for novices. All other differences were non-significant (see tables 9.25 and 9.26, as well as figure 9.21).

In order to compare the results between the two SC stimuli, Wilcoxon matched-pairs tests were used. The two stimuli differ significantly for almost all scores. For Text Order, all three types of scores are higher for L3_SC1/SC1 than for L5_SC3/SC3. For the Execution Order model, naive glocal scores are significantly lower for L3_SC1/SC1 than for L5_SC3/SC3, naive global scores are significantly higher. No difference was found between dynamic global scores (table 9.27).



(a) Naive glocal scores - novices

(b) Naive glocal scores - experts

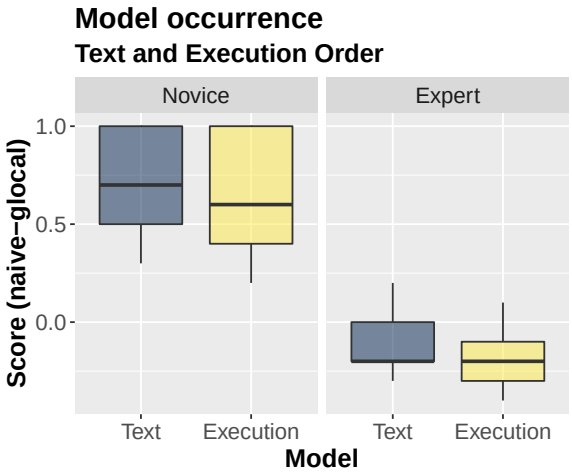
(c) Dynamic global scores - novices

(d) Dynamic global scores - experts

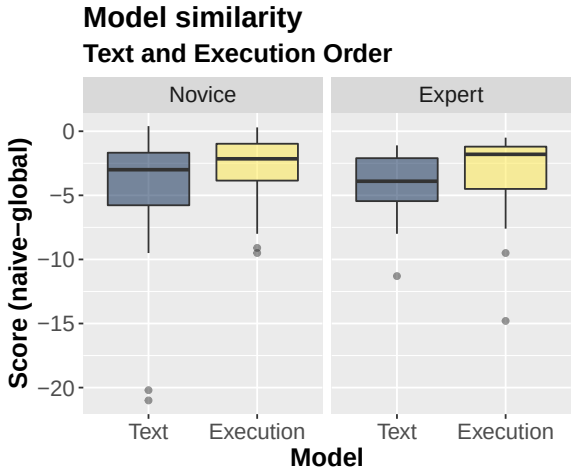
Figure 9.19: Direction of the difference in Text Order scores between NT and SC stimuli

Interpretation:
The three NT stimuli are comparable with regard to how much the Text Order sequence is present within the gaze and how similar the gaze is to the model sequence. Novices and experts are comparable in how much they read NT at least once in Text Order and mostly in how much their gaze resembles the model. In a number of NT trials, glocal scores of value '1' were obtained, meaning that the text was at least once read entirely according to Text Order. Glocal and dynamic global scores were predominately positive and higher than naive global scores. The higher dynamic global than naive global scores indicate that the texts were for the most part read more than once. The average number of times the Text Order model was repeated to achieve the highest dynamic global similarity was 6 [2..10] times. This finding aligns well with the participants knowing that they will be quizzed about the text and the fact that the texts were very short, so re-reading them in order to fully understand them and remember their gist for the upcoming comprehension question is easily done and does not take much time.

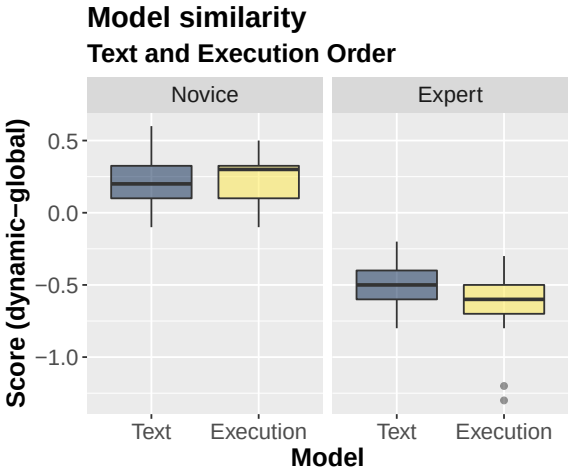
Novices follow the Text Order on lesson 1 SC as much as on NT. Glocal and dynamic global scores are almost exclusively positive, indicating that the novice participants read both text types in a rather linear way. This can be interpreted as transferring the customary predominantly linear reading strategy from NT to SC. However, the programs in the first lesson were rather linear, so they also might not elicit a radically different reading strategy than NT. For experts on the other hand, the Text Order model is significantly more present when reading NT than when reading SC and likewise the dynamic global similarity for Text Order is much higher for NT than for SC. Thus, experts read NT much more linearly than SC. For novices it is rather arbitrary whether the Text Order model receives higher scores on NT or on SC, while for experts the Text Order model is clearly a better fit for NT reading than for SC reading.



(a) Naive glocal scores



(b) Naive global scores



(c) Dynamic global scores

Figure 9.20: Text and Execution Order scores on SC stimuli

			L3_SC1/SC1	L5_SC3/SC3
Number of trials - novices			10	9
Number of trials - experts			12	13
Median naive glocal score - stimulus	Text	Order	-0.1 [-0.2..0.1]	-0.2 [-0.2..-0.1]
Median naive glocal score - novices	Text	Order	-0.1 [-0.2..0.1]	-0.2 [-0.2..-0.1]
Median naive glocal score - experts	Text	Order	-0.1 [-0.2..0.1]	-0.2 [-0.2..-0.2]
p-value novices vs. experts	Text	Order	0.893	0.260
Median naive glocal score - stimulus	Execution	Order	-0.3 [-0.4..-0.3]	-0.2 [-0.2..-0.1]
Median naive glocal score - novices	Execution	Order	-0.4 [-0.4..-0.3]	-0.1 [-0.2..-0.1]
Median naive glocal score - experts	Execution	Order	-0.3 [-0.4..-0.3]	-0.2 [-0.2..-0.1]
p-value novices vs. experts	Execution	Order	0.544	0.386

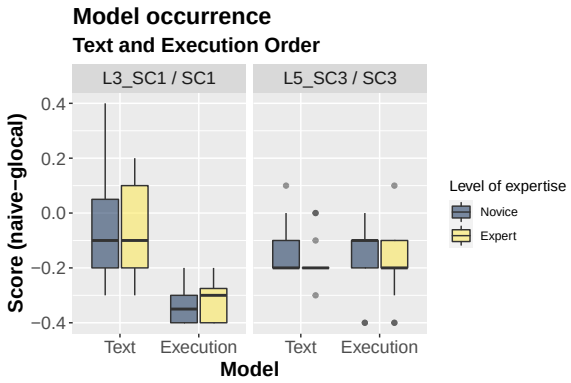
Table 9.25: Occurrence of Text and Execution Order sequences on the two programs viewed by both novices and experts

			L3_SC1/SC1	L5_SC3/SC3
Median global score - stimulus	Text Order	naive	-2.8 [-4.5..-1.5]	-4.2 [-5.6..-2.9]
Median global score - novices			-3 [-4.5..-1.7]	-4.4 [-6..-3.2]
Median global score - experts			-2.5 [-4.1..-1.6]	-3.9 [-5..-2.6]
p-value novices vs. experts			0.843	0.794
Median global score - stimulus		dynamic	-0.5 [-0.6..-0.3]	-0.6 [-0.7..-0.5]
Median global score - novices			-0.6 [-0.6..-0.4]	-0.6 [-0.7..-0.5]
Median global score - experts			-0.4 [-0.5..-0.4]	-0.6 [-0.7..-0.5]
p-value novices vs. experts			0.158	0.368
Median number of repetitions - stimulus			3 [3..5]	4 [3..5]
Median number of repetitions - novices			3 [3..4]	4 [4..5]
Median number of repetitions - experts			3 [3..5]	5 [3..5]
Median global score - stimulus	Execution Order	naive	-1.5 [-2.4..-0.8]	-5.5 [-7.4..-4]
Median global score - novices			-1.6 [-2.4..-0.9]	-6.2 [-7.9..-4.4]
Median global score - experts			-1.4 [-1.8..-0.9]	-5.1 [-6.8..-3.2]
p-value novices vs. experts			0.644	0.664
Median global score - stimulus		dynamic	-0.6 [-0.7..-0.5]	-0.6 [-0.7..-0.5]
Median global score - novices			-0.8 [-0.9..-0.7]	-0.6 [-0.7..-0.6]
Median global score - experts			-0.6 [-0.6..-0.5]	-0.6 [-0.7..-0.5]
p-value novices vs. experts			**0.002**	0.582
Median number of repetitions - stimulus			2 [1..3]	5 [4..7]
Median number of repetitions - novices			2 [1..2]	5 [4..6]
Median number of repetitions - experts			2 [2..3]	5 [4..7]

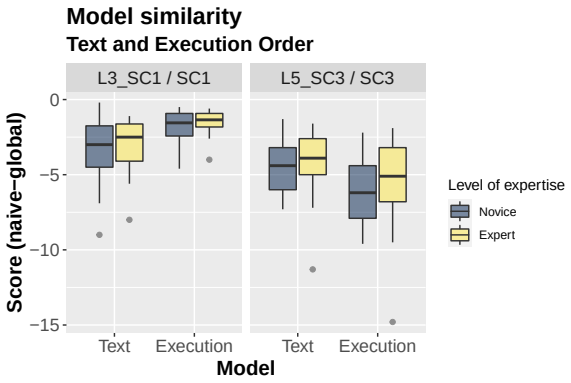
Table 9.26: Similarity to Text and Execution Order sequences on the two programs viewed by both novices and experts

	Text Order	Execution Order
Naive glocal	**0.045**	**0.001**
Naive global	**0.022**	**<0.001**
Dynamic global	**0.008**	0.736

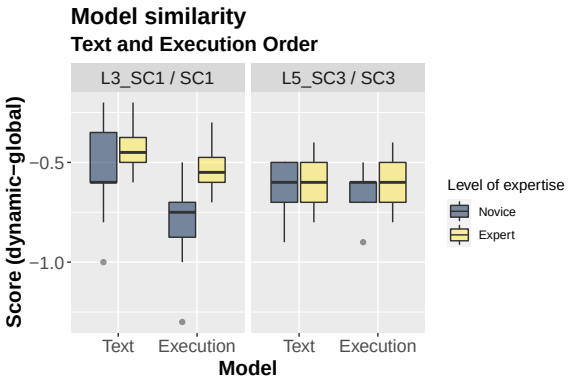
Table 9.27: Comparison of scores between the SC stimuli L3_SC1/SC1 and L5_SC3/SC3



(a) Naive glocal scores



(b) Naive global scores



(c) Dynamic global scores

Figure 9.21: Text and Execution Order scores for the two programs viewed by both novices and experts

Novices partly read the SCs at least once completely in Text Order (highest glocal score of '1'), experts on the other hand never went over a SC even close to linearly (highest glocal score of '0.2'). Considering that the SCs for novices consisted of 4 and 6 lines, those for experts of 18 and 22 lines, this is not surprising, since it is perfectly feasible to read a few lines without going back or skipping, but for about 20 lines that would require a lot of concentration and control over one's eye movements, even when reading a rather simple text. Similarly, Day [2010, 401] states that the global similarity scores obtained in an eye tracking experiment on decision making strategies were lower than expected and surmises that this is partly because participants were not able to execute a search sequence without occasional diversion, e.g. by making regressions during reading.

When comparing the Text and Execution Order scores on SC stimuli, for novices the naive glocal and dynamic global scores for both models are comparable, while the naive global scores for Text Order were significantly lower than for Execution Order. For experts, naive glocal scores are significantly higher for Text Order, while naive global scores are significantly higher for Execution Order. No difference was found for dynamic global similarity. With regard to naive glocal alignments, no difference between Text and Execution Order scores means that the gaze sequences contain episodes that follow Text Order as well as episodes that comparably follow Execution Order. Reading a program at least once in Text Order can reasonably be expected of both novices and experts. It can be interpreted as applying the linear NT reading approach to SC, but it can also very well be a scan, i.e. an initial complete reading through the program before focusing on its details [Sharif et al., 2012], [Uwano et al., 2006], [Uwano et al., 2007]. Since alignment algorithms start to align sequences from their ends and the naive glocal approach stops at the first optimal solution, it is not possible to look for Text Order episodes at the beginning of a sequence without further adapting the alignment procedure, so at this point it cannot be verified whether a Text Order episode occurs at the beginning of a gaze sequence.

Experts having no significant difference between Text and Execution Order scores might at first

suggest that both models comparably fit to the gaze. However, while for novices the lengths of the two model sequences per text were very similar, in the expert SCs, Execution Order sequences are partly considerably longer than Text Order sequences. Regardless of normalization for model length, long sequences have a lesser chance of being followed closely than short ones, since there is much more potential for deviation as it is difficult to adhere one's gaze to a certain pattern for longer periods of time. Besides, jumping to the next line according to Text Order is easy, while locating the next line according to Execution Order might require some orientating. So even though someone wanted to move the gaze directly to the line that is executed next, it can be necessary to look around first in order to find it, and thus the model sequence is interrupted. In the short programs from lesson 1, this is hardly relevant, but for the longer expert codes, it might very well dilute the order in which participants look at the lines. The potential advantage that a longer model sequence might contain more matching items and produce less gaps is mostly leveled out by normalization, but the disadvantage of being more difficult to follow remains. Thus, if the length of one model sequence is considerably longer than the other or one sequence necessitates much more jumping, the comparison of the resulting alignment scores is not as direct as if both sequences are of similar length and character. This issue arises with glocal and global alignments alike. In such cases the score of the model behavior represented by the long sequence should be regarded differently than the score of the short model sequence. However, there is no straightforward relation of how much deviation from the model can be accounted for solely by model length, so the results for these stimuli need to be interpreted more carefully. SC2 is inspected as an example. It has the greatest difference between the lengths of the Text Order sequence (22 lines) and Execution Order sequence (52 lines) and the two model sequences are the least similar. Here, both models are comparably present. Average naive glocal scores are -0.1 [-0.2..0] for Text Order and -0.1 [-0.2..0.1] for Execution Order. The scores for Execution Order are slightly better than for Text Order, but the difference is not significant. The finding that the Execution Order model is present as much as the Text Order sequence despite being 30 lines longer can be interpreted in favor of the Execution Order model. The notion that Execution Order is actually an overall better fit for the gaze than Text Order is corroborated by the global similarity scores. Naive global scores for Execution Order are even significantly higher than for Text Order (average Text Order score: -4.8 [-6.0..-4.2], average Execution Order score: -1.2 [-1.6..-1.1]), dynamic global scores for both models are comparable (average Text Order score: -0.6 [-0.7..-0.4], average Execution Order score: -0.6 [-0.8..-0.5]). When repeating these comparisons with the not normalized scores, the results of the comparisons remain the same despite the huge length difference, i.e. glocal and dynamic global scores are comparable between models, and the naive global score is significantly higher for Execution Order. An additional factor is that the recording environment for experts was less optimal than for novices. Thus, the data quality is potentially lower for experts, resulting in longer and less homogeneous collapsed AOI sequences, which makes the results for experts less robust.

In summary, it can be reliably concluded that at the beginning of the programming course, the novices read the presented NTs and SCs rather linearly. For experts however, Text Order better describes the NT reading behavior than the SC reading approach. Whether Execution Order better accounts for their SC reading behavior cannot be unequivocally be determined, but the comparable Text Order and Execution Order scores despite the much greater lengths of Execution Order sequences suggests that experts actually focus more on Execution than on Text Order. Another general finding is, that naive global scores are generally lower than glocal and dynamic scores, implying that the SC stimuli were mostly read several times.

When comparing novices and experts on the two stimuli viewed by both groups, mostly no differences were found between the scores of novices and experts. However, the dynamic global Execution Order score on program L3_SC1/SC1 showed that experts read the program much more in Execution Order than novices. Otherwise, model occurrence and similarity was comparable between groups. Consequently, by the time novices reached lesson 3, novices' and experts' gaze mostly fits comparably to the two models. However, at least the comparable global scores for Text Order might result from different behaviors. For experts low agreement with Text Order is probably due to them not reading programs in a top-to-bottom manner, but to heading for certain code areas rather purposefully. Novices on the other hand might thrash around more in search of comprehension clues and therefore have lower similarity to the Text Order model, even if the general reading approach is rather linear. As before, naive global scores are much lower than glocal and dynamic global scores. Furthermore, scores mostly differ significantly between the two SC stimuli. For the two programs that were studied, how much a model sequence fits to the gaze seems to be more influenced by characteristics of the stimulus than by level of expertise.

9.4 Trial-based measures

9.4.1 Trial duration

In total, the EMCR trials on NT and SC amount to a duration of 2 hours and 40 minutes, whereat SC data accounts for the larger part (see table 9.28).

Trials	Total duration (min)
All	160
NT	74
SC	87
SC - novices	40
SC - experts	47

Table 9.28: Sum of trial durations for different sets of trials

For NT, 72 trial durations were analyzed, 27 for novices and 45 for experts. The durations show a log-normal distribution, so parametric tests can be employed. Log-transformed values were used for testing. The mean duration over all NTs is 61 sec (SD=32). A two-way ANOVA showed no difference between the trial durations of the three texts (p=0.121), nor between novices and experts (p=0.399). Table 9.29 and figure 9.22 detail the trial durations per text.

	NT1	NT2	NT3
Number of trials - novices	10	9	8
Number of trials - experts	15	15	15
Mean duration - stimulus	54 sec (SD=33)	68 sec (SD=30)	62 sec (SD=31)
Mean duration - novices	53 sec (SD=36)	65 sec (SD=31)	58 sec (SD=26)
Mean duration - experts	54 sec (SD=32)	70 sec (SD=30)	65 sec (SD=35)

Table 9.29: Trial durations for NT stimuli

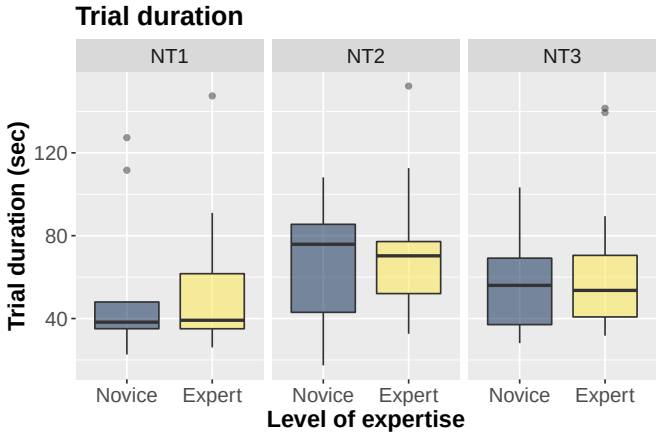


Figure 9.22: NT trial durations per group of expertise

RQ2:
For the two SCs shown to both groups of expertise, 45 trial durations were recorded, 19 for novices and 26 for experts. Again, the durations follow a log-normal distribution, so parametric tests were employed with log-transformed durations. On the first SC the mean duration for novices is 77 sec (SD=27), for experts 64 sec (SD=36), on the second SC the mean duration for novices is 100 sec (SD=24), for experts 75 sec (SD=41). Thus, on average, novices needed more time to read the two programs. However, using independent samples t-tests, only the difference for the second SC proved to be significant, see table 9.30 and figure 9.23. Furthermore, even though the average duration for L5_SC3/SC3 (85 sec, SD=36) was longer than for L3_SC1/SC1 (70 sec, SD=33), the difference is not significant (p=0.136).

	L3_SC1/SC1	**L5_SC3/SC3**
Number of trials - novices	10	9
Number of trials - experts	13	13
Mean duration - stimulus	70 sec (SD=33)	85 sec (SD=36)
Mean duration - novices	77 sec (SD=27)	100 sec (SD=24)
Mean duration - experts	64 sec (SD=36)	75 sec (SD=41)
p-value novices vs. experts	0.202	**0.034**

Table 9.30: Trial durations for the two programs viewed by both novices and experts

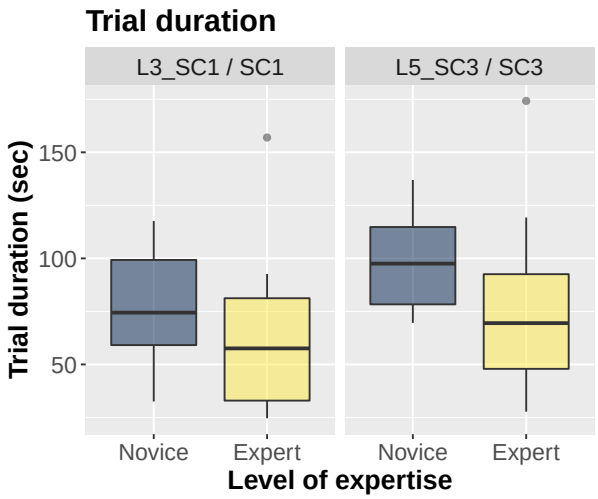


Figure 9.23: Trial durations of the two programs viewed by both novices and experts

Interpretation

Trial durations on NT are highly comparable between texts as well as between the two expertise groups. Consequently, regarding the time needed to read the NTs, the participating novice and expert programmers are similar readers. On the two SCs that were presented to both groups, the average trial duration of novices was longer on both programs, but only the difference for the second and more complicated program L5_SC3/SC3 is significant.

9.4.2 Correctness of comprehension question

Overall, 76 NT correctness scores were obtained, 29 for novices and 47 for experts. The obtained scores are not normally distributed, so non-parametric tests were used for all comparisons. The vast majority of participants answered the NT comprehension questions completely correctly (79%), resulting in an overall median correctness score of 1 [1..1]. Likewise, the median score for each NT is 1, with an interquartile range of [1..1] for NT1 and NT2, and [0.5..1] for NT3. Using a Kruskal-Wallis test, no significant difference was found between the correctness scores on the three NTs (p=0.605). Furthermore a Mann-Whitney test showed no difference in NT comprehension between programming novices and experts (p=0.646), see table 9.31.

	NT1	**NT2**	**NT3**
Number of scores - novices	10	10	9
Number of scores - experts	16	15	16
Median score - stimulus	1 [1..1]	1 [1..1]	1 [0.5..1]
Median score - novices	1 [1..1]	1 [0.6..1]	1 [1..1]
Median score - experts	1 [1..1]	1 [1..1]	1 [0.4..1]

Table 9.31: Scores for NT comprehension questions

RQ2:
For the two SCs that were read by both groups of expertise, 44 comprehension scores were recorded, 18 for novices and 26 for experts. Because of a software failure, one novice score for L3_SC1 is missing even though there is gaze data for that trial. The scores are not normally distributed, so non-parametric tests were used. The answers to the comprehension question for L3_SC1/SC1 are completely correct 15 out of 22 times (68%) with a median correctness score of 1 [0.5..1.0]. For L5_SC3/SC3 11 of 22 answers were correct (50%), with a median correctness score of 0.75 [0.5..1.0]. Using Mann-Whitney tests, no significant difference was found between the comprehension scores of novices and experts on either of the two programs (table 9.32). Even though the median score for L5_SC3/SC3 (0.75 [0.5..1]) is lower than for L3_SC1/SC1 (1 [0.5..1]), the difference between both programs is not significant (p=0.438).

	L3_SC1/SC1	**L5_SC3/SC3**
Number of scores - novices	9	9
Number of scores - experts	13	13
Median score - stimulus	1 [0.5..1]	0.75 [0.5..1]
Median score - novices	1 [0.5..1]	0.5 [0.5..1]
Median score - experts	1 [0.5..1]	1 [0.5..1]
p-value novices vs. experts	0.968	0.119

Table 9.32: Scores for SC comprehension questions on the two programs viewed by both novices and experts

Interpretation

Almost all participants answered the NT comprehension questions correctly and no difference could be found in comprehension scores between the three stimuli nor between programming novices and experts. Consequently, none of the NTs was too difficult to understand for the participants, and novices and experts do not differ in their ability to comprehend the presented NTs.

No difference was found between comprehension scores of novices and experts on SC. So while novices partly needed more time to understand the programs, their performance on the comprehension questions was comparable to that of the experts. The median scores of 1 [0.5..1] and 0.5 [0.5..1] for novices, suggest that the difficulty of the programs was adequate for the state of knowledge of the novices at

the respective point in their programming course, even though they struggled slightly more with the later program. Experts understood both programs well (median correctness score of 1 [0.5..1] for both stimuli). The reason, why the novices partly performed similarly well with regard to time needed to read the programs and comprehension correctness, is most likely that they just finished the lesson in the course that covered the relevant concepts. Hence they had just practiced programming with comparable programs and apparently grasped them well. Besides, when scoring the summary task the participant's level of expertise was taken into account. The longer average trial duration and lower comprehension score for L5_SC3/SC3 indicate that this program is more difficult than L3_SC1/SC1, however the differences are not statistically verifiable.

9.5 Threats to validity

Careful steps were taken to ensure the validity of data and results. Nevertheless, threats may arise from the study design, the recording environment, analysis procedures, as well as the participants.

Internal validity
Gaze data was recorded with an SMI RED-m eye tracker, a reliable eye tracking device with high precision and accuracy, which is aimed at use in scientific research and supposed to work robustly for a wide range of participants and recording situations [Mele & Federici, 2012], [SensoMotoric Instruments, 2016a]. When designing the stimuli, it was taken into account that eye tracking devices are not completely exact and the eye tracker was frequently recalibrated. Nevertheless, there is a limit to the data quality that can be achieved. The data was recorded in natural daylight, so the light conditions were not optimal. Additionally, to keep the situation as ecologically valid as possible participants were not restrained. The RED-m compensates for head movements up to a certain degree, but they are still a source for errors. While almost all novice data was recorded in the same computer lab, most experts conducted the study at their workplace. On the one hand, this increased the validity of the recording situation. On the other hand, the varying environments were partly less ideal for eye tracking, so the quality of the expert data might be lower than for novices. Findings from the evaluation of the error correction method suggest that the experts' recordings indeed contained a greater error than those of the novices. A further issue is that the SC stimuli presented to novices in lesson 3 and 5, and to experts were longer and more complicated than the NTs and the SCs in lesson 1. The trials on these stimuli took more time and the text spread closer to the margins of the screen, so their quality may be potentially lower than for NT and lesson 1. The applied error correction addressed this issue and the recordings used when gaze location was crucial were selected carefully. Therefore, the data included in the analyses based on location should all be of sufficient quality for the drawn results, but it cannot be ruled out that data from some trials is of lower quality than others.

The recording situation was designed to be very ecologically valid. The RED-m eye tracker is a small non-intrusive device fastened to the frame underneath the display. Participants were not attached to any equipment or restricted in any way, so the recording was not very noticeable. Especially the novices claimed to hardly be aware of the eye tracker due to the repeated recordings. In addition, the data was not collected in a special lab. The novices were recorded almost exclusively in the same computer room they attended the programming course in, so they were very familiar with it. Besides, it is a regular classroom and not an artificial setup. Experts were mostly recorded at their workplace using their own display and keyboard, so they were also very familiar with the environment. Nevertheless, participants were aware that their gaze is being recorded. This and knowing that they are required to answer a comprehension question might have afflicted some persons, even though it was emphasized that the recording is not any kind of exam and that they cannot do anything wrong. Furthermore, the researcher conducting the recording did not look over the participants' shoulders while their gaze was recorded, but gave them space to work on the tasks without being under constant observation to reduce the effect of participants adapting their behavior because they feel watched [Sharafi et al., 2015b, 100], [Sharafi et al., 2020, 3155]. Overall, the setup was very unobtrusive, nevertheless, some participants might have shown a different reading behavior than they would in a completely natural situation. Furthermore, all processing steps were automated as much as possible to ensure exact and objective results, e.g. AOI coordinates were calculated directly from the stimulus image.

External validity
Eye movements are heavily guided by the stimulus and task at hand. The languages used in the EMCR study were chosen for their representativeness and the employed tasks represent common activities during programming, both when learning a programming language as well as during professional software development. However, the tasks were not distinguished during analysis. The English texts were taken from a reading test and the programs were adapted from another study or actual teaching material in order to generate valid stimuli. Also, they include essential concepts like loops and conditions. Nevertheless, the specific texts and associated comprehension tasks might include unfavorable aspects that limit the generalizability of some findings. For example, program L5_SC3/SC3 contains a method called `accelerate` which also allows to decrease the speed of a vehicle by accepting negative acceleration values. This pitfall was incorporated into an otherwise simple program so it is not too easy for the experts, yet only contains basic constructs so that novices can understand it. However, such a "trick question" is very artificial and possibly resulted in confusing some participants. Two novice programs were written in pseudocode, which also could be a factor in some of the analyses. All texts are rather short (maximum of 22 lines) to fit the screen without scrolling. While such short programs are typical for programming education, they are somewhat artificial for experts. Since this is a common constraint in eye tracking studies on software engineering [Obaidellah et al., 2018, 33], [Sharafi et al., 2015b, 100], [Sharafi et al., 2020, 3144], the design in any case allows to integrate the findings with many other results obtained from small programs.

Furthermore the formatting can influence the results. At least the expert programmers who participated in the study are used to a certain color scheme for syntax highlighting, which differs among participants. In order to establish the same condition for all participants, the stimulus programs were presented in plain format without any color. Using C#.Net as programming language, Beelders & Plessis [2016a] and Beelders & Plessis [2016b] found that displaying code with and without syntax highlighting to IT students had no significant effect on the number of fixations, fixation durations, and number of regressions. Thus, it is presumed that the absence of syntax highlighting is a negligible issue. To facilitate the mapping of gaze to the stimulus, white spaces were added where possible, but only when it did not disturb the structure of the text. Nevertheless, it might still have irritated someone. The order in which the texts were presented was randomized to avoid effects from always showing a text at a certain point in the experiment. Yet, despite the careful design, using another set of stimuli will yield at least partially different findings.

The EMCR study includes 26 participants, which is slightly more than the typical number of participants in such eye tracking studies, and even features professional programmers instead of students, a plus compared to some other studies [Obaidellah et al., 2018, 17-19], [Sharafi et al., 2015b, 90,92,101], [Sharafi et al., 2020, 3145,3146]. Despite the adequate participants, it cannot be ruled out that some findings are specific to the studied individuals and not representative. Some eye physiologies are more suitable for being recorded by an eye tracker than others, so data quality varies among participants. Besides, even though all participants were proficient in English, none were native speakers, which has to be considered for the findings on NT reading.

Construct validity
Like most research using eye movement data, the EMCR study largely draws on the widely accepted assumption that gaze and attention are strongly linked. Furthermore, eye tracking is intensely based on software [Hornof & Halverson, 2002, 593,594]. The gaze location is calculated rather than measured with a device like a ruler. On one hand, a computer is more objective than a human, on the other hand, software is written by humans and not without fail. Optic artifacts, i.e. impossible eye movements found in the data [Holmqvist et al., 2011, 33,34], are a strong reminder of that.

Furthermore, the detection of oculomotor events can be problematic. The definitions of many events are somewhat ambiguous, and there is no complete consensus about what exactly constitutes a fixation or a saccade. For instance, some attribute glissades to fixations others to saccades or consider them noise. Even for humans, it is partly difficult to decide whether certain samples belong to a fixation or another type of event. Furthermore, the events used in eye tracking research are usually calculated entities and depend on the used detection algorithm and parameters. So the same raw data will yield somewhat different events when processed with another detection approach. The employed I-DT was carefully chosen and adapted to the EMCR data, nevertheless not all samples might be classified correctly. Also, the results obtained with the here computed events are probably not entirely comparable to others from a different detection approach. However, the EMCR analysis results are based on fixations and saccades

that were calculated with exactly the same algorithm and parameters, so these events were attained consistently and objectively. Another point to consider is that the sample location is relevant when identifying fixations, but the procedure to correct spatial errors is only applied after fixation detection. Thus, determining fixations is susceptible to errors. The subsequent correction rectifies this obstacle to a great extent, but cannot oust it completely. Besides, the error correction itself is a modification to the data, albeit a well-founded one.

The indicated threats to validity are mostly typical for eye tracking studies and were addressed carefully as well as taken into account during analysis. Despite the challenges associated with eye tracking, gaze data provides very rich and detailed information about code reading and comprehension that is not accessible by other instruments like think-aloud or interviews and the EMCR data represents a good basis for the conducted analyses.

10 Conclusion

10.1 Synopsis

The EMCR study was conducted with the objective to advance the research methodology in the domain of program comprehension and to deepen the knowledge thereof. A methodological framework was presented that allows to analyze eye movements during program comprehension and two exemplary research questions were answered regarding code reading, an observable component of comprehension.

10.1.1 Natural-language text reading

In addition to code reading, data on the participants' NT reading behavior was collected in order to assess whether findings on NT are largely transferable to SC reading and to have a baseline for interpreting the SC results.

Overall, the EMCR participants show a regular NT reading behavior on the English texts, even though they were not native speakers. Their fixation durations when reading NT reside within a normal range and they predominantly read linearly top-to-bottom and left-to-right. The three stimulus texts were understood correctly by the majority of participants and are mostly interchangeable as they induce comparable results for almost all tested measures. Solely the element coverage is slightly higher on NT2 than on the other two texts (median=100% instead of 99%). Even the times needed for comprehension are comparable. Thus, it is of hardly any consequence that for some participants not all NT stimuli are available for analysis. The participants of the novice and expert programmer groups exhibit for the most part comparable NT reading behaviors. Fixation duration proved to be highly variable and idiosyncratic, so the identified differences for this measure are attributable to individual traits. The few differences found between the participating novices and experts when reading NT were taken into account when interpreting the results of the respective measure with regard to the research questions.

10.1.2 Research question 1

Is reading behavior different between natural-language text and source code?
If so, is the difference already present in novices?

Fixation duration is a highly individual measure and can vary substantially over a trial. Reading source code instead of natural-language text did not radically change the participants' fixation durations. For novices the distribution of fixation durations on SC differs significantly from that on NT, while the average fixation duration does not. For experts however, both the distribution and average fixation duration are comparable on NT and SC. All participants were at least advanced, but not native English speakers, so the foreign language is a factor when comparing fixation durations on NT and SC. With regard to AOI coverage however, the NT and SC reading behaviors are very different. Both novices and experts looked at significantly more lines and elements of the English texts than the SCs. Likewise, both groups of expertise

have a much higher proportion of linear reading on line- and element-level for the NT stimuli than for SC. Thus, locally both novices and experts read SC much less linearly than NT. Nevertheless, when analyzing the overall reading approach, novices read NT and SC comparably according to Text Order. Experts on the other hand follow Text Order on NT much more than on SC. Text Order and Execution Order mostly fit equally to the novices' gaze when reading SC. For experts no definite conclusion can be drawn about which model better characterizes their gaze during code reading when comparing Text and Execution Order scores on SC. However, there are plausible indications that overall Execution Order is at least partly a better fit.

SC is read differently than NT in several aspects. Differences in the reading behavior were already found for early novices, for expert programmers they are even more pronounced. Consequently, the two types of text do indeed induce different reading behaviors, it has to be thoroughly examined to which extent measures and findings from NT reading apply to code reading as well. Table 10.1 summarizes the most prominent results.

		Measure	Novices	Experts
single-event-based		Distribution of fixation duration	NT ≠ SC	NT = SC
		Median fixation duration	NT = SC	NT = SC
		AOI coverage (line-level)	NT > SC	NT > SC
		AOI coverage (element-level)	NT > SC	NT > SC
event-sequence-based		Proportion of linear reading (line-level)	NT > SC	NT > SC
		Proportion of linear reading (element-level)	NT > SC	NT > SC
		Model occurrence - Text Order	NT = SC	NT > SC
		Dynamic model similarity - Text Order	NT = SC	NT > SC

Table 10.1: Summary of the most notable results for research question 1. AOI coverage and proportion of linear reading differ between NT and SC irrespective of programming expertise. The Text Order model fits equally well to the novices' reading approach on NT and SC, while for experts it matches the NT reading more than the SC reading. Comparisons that showed statistically significant differences between NT and SC reading are highlighted.

10.1.3 Research question 2

Do novices exhibit a different code reading behavior than experts?

The distribution of fixation durations differs between novices and experts on both tested programs. For the first SC (L3_SC1/SC1) average fixation duration and number of fixations were comparable for both groups, while on the second program (L5_SC3/SC3) novices had significantly higher average fixation durations, but similar number of fixations. Median and distribution of fixation duration are comparable between the two SCs. Novices and experts were also found to have different distributions of saccadic amplitude on both programs, while average saccadic amplitude as well as scanpath length are comparable. Similarly, AOI coverage was mostly comparable between novices and experts. Solely on L3_SC1/SC1 novices looked at more lines than experts. In contrast, the time and number of fixations until the participants visit the `main`-method for the first time are much higher for novices than for experts. The differences occur strongly in both programs. Experts visit the `main` very early when reading the programs, while novices spend a lot of time on the code above it. Given that the `main`-method was located at the end of the programs and experts usually looked at much less than half of the lines above `main` before their first visit there, this finding strongly suggests that experts purposefully went for this method. Once they reached the `main`, the dwell does not differ between the two groups of expertise.

On program L3_SC1/SC1 novices and experts exhibit a comparable proportion of linear reading, though novices tend to read lines in a more linear manner than experts. The later program L5_SC3/SC3 was read significantly more linearly by novices than by experts - both on line and element level. Novices already read NT more linearly on element-level than experts, so the difference on the finer level might be partly the result of the individual reading behavior of the studied persons and not only of programming expertise. With regard to occurrence of and similarity to the two models Text Order and Execution Order, hardly any differences were found between the two groups of expertise. Only on L3_SC1/SC1, experts followed the dynamic global Execution Order significantly more than novices. On average novices needed more time to read and understand the programs than experts, but the difference is only significant for the second more complicated SC. With regard to comprehension, both novices and experts reached similar scores given their expertise.

Novices and experts were found to be similar for a number of measures. They mostly differ with regard to the order in which they read SC. Most notably, experts proceed to look at the `main`-method very early after starting to read a program, while novices lack this purposeful behavior. Furthermore, the novices partly display a more linear reading behavior than the experts. A summary of important results is provided in table 10.2.

	Measure	L3_SC1/SC1	L5_SC3/SC3
single-event-based	Distribution of fixation duration	novices ≠ experts	novices ≠ experts
	Median fixation duration	novices = experts	novices > experts
	Distribution of saccadic amplitude	novices ≠ experts	novices ≠ experts
	Median saccadic amplitude	novices = experts	novices = experts
	Time till first visit to `main`	novices > experts	novices > experts
	Number of fixations till first visit to `main`	novices > experts	novices > experts
	Dwell time of first visit to `main`	novices = experts	novices = experts
	Number of fixations in first visit to `main`	novices = experts	novices = experts
event-sequence-based	Proportion of linear reading (line-level)	novices = experts	novices > experts
	Proportion of linear reading (element-level)	novices = experts	novices > experts
	Model occurrence - Text Order	novices = experts	novices = experts
	Dynamic model similarity - Text Order	novices = experts	novices = experts
	Model occurrence - Execution Order	novices = experts	novices = experts
	Dynamic model similarity - Execution Order	novices < experts	novices = experts

Table 10.2: Summary of the most notable results for research question 2. The distributions of fixation durations and saccadic amplitudes differ between novices and experts on both programs. The time and number of fixations until the `main`-method is visited for the first time are the most prominent differences between the two groups of expertise. Moreover, novices tend to read the programs more linearly than experts. Comparisons that showed statistically significant differences between novices and experts are highlighted.

10.2 Reflection on methods and analysis measures

Preparing gaze data for analysis is a crucial part of the EMCR study. The central steps are determining fixations and saccades from raw gaze data, as well as the correction of spatial errors. Both procedures were developed specifically for the EMCR data, but can be applied to a multitude of other gaze data not just from programming- or reading-related stimuli.

Fixations were detected with an adapted identification by dispersion threshold algorithm suitable for the two sampling rates present in the EMCR data. The specific implementation sets the dispersion threshold in milliseconds instead of number of samples, uses the maximum distance between samples as dispersion measure, allows a certain amount of samples outside the specified dispersion threshold, and introduces a maximum amount of time that can be missing between samples within a fixation. These adaptions help to overcome several of the I-DT's usual drawbacks and make it more robust even for noisy data. Furthermore, it is based on a very palpable principle and has a wide applicability, as it allows to process raw gaze data that was recorded with various sampling rates using the same approach since raw data with a high sampling rate can be down-sampled. Velocity-based event detection on the other hand has a narrower scope, since it is problematic for data from slower eye trackers. In addition, the I-DT can be adjusted to the data at hand by its two parameters. On the downside, the I-DT might misclassify samples under certain conditions, e.g. low precision of the data. The instantiated adaptions reduce this issue, but do not remove it completely. Besides, some of the modifications make the algorithm computationally intensive and unusable for on-the-fly event detection, which is irrelevant for the EMCR study, but might be a factor for others. Evaluating the adapted I-DT as well as the chosen parameter settings showed that they are a good choice for the EMCR data.

Drawing from other methods for correcting spatial errors, a novel approach was developed that takes the distribution of AOIs on the stimulus into account. Besides adopting the common steps for preparing stimuli to lessen the impact of errors, like adding extra margins where possible, AOIs in less dense parts of the stimuli were adapted to allow for greater disparities. Evaluating correction methods itself poses a challenge, so several options were applied. Comparing the fixations to reference locations, manually corrected data, and artificial data all provides valuable information about the suitability of a correction method, as well as data quality, but only their combination allows to draw a more comprehensive picture. The evaluation showed that the correction procedures considerably improved data quality. However, the performance varies depending on the data that is to be processed. The large amount of parameters, that are tested to find an optimal solution to the correction, results in a very long computing time. To increase the applicability of the correction procedures, a heuristic is needed to reduce the parameter search space to a more reasonable size. Moreover, automatically corrected data should still be inspected for plausibility by at least one person, better yet two as practiced for the EMCR data.

Further methodological aspects include complementing gaze data with trial duration and correctness of the comprehension task. In addition, taking interview data from the expert programmers into account facilitated defining the Execution Order model. Combining different types of data yielded rich information and has a lot of potential for further development. Furthermore, as many steps as possible were automated, e.g. determining AOIs algorithmically, establishing a high degree of objectivity and allowing to process larger amounts of data.

Fixation duration, number of fixations, saccadic amplitude, and the associated measures like scanpath length were found to be highly idiosyncratic and not very distinctive for either research question, which is in itself a relevant finding. Fixation duration and saccadic amplitude can vary substantially over the course of a trial and aggregated values partly conceal differences found in their distribution. Yet, fixations and saccades are very basic events and provide the foundation for other more complex measures, and they yield vital information about the quality of the collected data. Besides, they are very universal in eye movement research, so applying them allows to relate the data to a multitude of other studies and findings. Especially duration and number of fixations as well as their variations are very often used in eye tracking studies about programming [Obaidellah et al., 2018, 27-30], Sharafi et al. [2015a]. Moreover, for other questions concerning program comprehension studying these elementary measures might very well result in more consequential findings. Thus, such measures should also be examined and reported in future studies when possible.

AOI coverage differed significantly between NT and SC stimuli, so it is a very distinguishing measure for the text type. For level of expertise on the other hand, it did not allow to differentiate between novice

and expert code reading behavior. However, only the proportion of AOIs was studied, not which AOIs were looked at. Thus it can very well be that experts and novices do focus on a similar amount of AOIs, but not on the same ones, e.g. different kinds of lexical elements or other sections of the program might be of interest to each group as illustrated by Crosby & Stelovsky [1990]. Since AOIs are not text-specific and can be created on varying levels of abstraction, AOI coverage can be applied for many questions and stimuli, e.g. understanding UML diagrams.

The first visit to the `main`-method can only be studied for programming languages which include this concept. For Java, a language widely used in programming education as well as in industry, the time and number of fixations until the `main` was looked at for the first time, strongly distinguishes novices from experts. The first dwell on `main` however was comparable for both levels of expertise. Consequently, the point in time in which the `main` is accessed is a key difference between novices and experts. Since the contrast was so compelling for Java, the first visit to `main` or a comparable method is definitive worth studying for other languages to ascertain that the effect is representative.

The proportion of linear reading proved to be influenced by text type. Both novices and experts read the English texts more linearly than the Java programs. This can be explained by the non-linear nature of the programs as well as by their difficulty, which might have elicited a higher portion of regressions. On SC novices have the tendency to read more linearly than experts. Consequently, this measure yields valuable insights with regard to stimulus type as well as expertise, and it is applicable to other research questions as it captures sequential information on different levels of abstraction. Furthermore it can easily be adapted for other stimuli, like programs in the languages Snap! or Scratch.

Aligning gaze with different models is a useful way of analyzing gaze data and testing how much a certain model accounts for the observed behavior. This is not limited to reading. Glocal alignments allow to detect episodes in the gaze that resemble a model sequence and open the possibility to divide the gaze into consecutive phases. With naive glocal alignments the gaze can be examined for the presence of different models, which might occur at some point, e.g. an episode of following the Execution Order. The approach used here can be adapted to look for parts of the model, e.g. only the execution of the `main`-method or the initial scan of the complete program. Additionally, with dynamic glocal alignments, episodes of repeatedly following a model order can be identified, even when the model is not a good fit for the overall strategy. Global alignments determine the general similarity between gaze and model. When the gaze contains more than one episode of looking at a stimulus in a certain way, naive global alignments are not a very suitable instrument to evaluate similarity between gaze and model. In order to assess the overall similarity of gaze to a model, dynamic global alignments are more appropriate, especially since the procedure starts with the unrepeated model. Thus, if the text (or any stimulus for that matter) is only looked at once according to model, the dynamic alignment will not repeat the model and align the gaze with just a single model instance. In addition to the types of alignment used in the EMCR analysis, local alignments can be employed to find sub-sequences that are highly similar between model and gaze or between participants. Such patterns can also be seen as episodes in the overall reading approach and serve as basis for developing further models to characterize code reading.

Alignment scores are highly affected by model length. Thus in order to fully tap the potential of this analysis instrument when comparing model behaviors, it might be worth considering to construct the stimuli in a way that the lengths of the model sequences are very similar while the sequences themselves are as different as possible. For SC this could be achieved by using loops, method calls, and conditions to cause the execution to jump to different locations in the program other than the next line, and simultaneously include code that will not necessarily be executed, e.g. an `else`-branch, to balance the length of Text and Execution Order. Finally, alignments can be used on artificial as well as on actual gaze data and for all purposes of a similarity measure suggested by Mathôt et al. [2012, 1,2]: finding differences in the gaze from pre-defined groups, e.g. novices and experts; finding clusters and patterns within gaze data; diagnosing to which group a given gaze sequence is most similar; and analyzing within- versus between-participant similarity, i.e. is the SC reading behavior from one person when re-reading a program (or even when reading different programs) more similar than between participants.

Trial duration equals the time needed to read the text and can be used without eye tracking. The same applies to the correctness of the comprehension question. Consequently, these measures have a broad field of application irrespective of eye movements and programming and allow to relate the findings to many other studies. They yield additional information about the participants' comprehension and still provide data even when the gaze recording fails or has insufficient quality.

10.3 Discussion and future work

Studying eye movements during code reading proved to be worthwhile the effort. The developed methodological framework reinforced the findings that source code elicits a different reading behavior than natural-language text and that proficient code reading requires its own specific approach.

It is usually taken for granted that novice programmers are trained readers. Consequently, programming instruction often puts only limited effort into further developing code reading skills, without realizing that the previously acquired reading behavior might partly be more of a hindrance than helpful. Skilled reading comprises more than just decoding written words. Various mental operations take place during reading to process and filter the visual information, and to determine where to direct the eyes next, thus reading and understanding are highly intertwined. The ability to understand given code is recurrently described as a distinct skill, partly even as precursor to other programming-related skills [Lister et al., 2004], [Lopez et al., 2008], [Venables et al., 2009], [Xie et al., 2018], [Xie et al., 2019]. It comprises understanding the syntax and tracing the execution, but also inferring the algorithmic idea or general purpose of the code. Simply transferring the chiefly top-to-bottom reading approach from natural-language text to source code seems not to be adequate. Novices lack the purposeful visual behavior found in experienced programmers, so expertise is in part reflected in the order in which programmers move their eyes over the code and presumably, there are different reading strategies associated with development, debugging, or mere comprehension.

In addition to the reading order, natural-language and program text differ in how much of the words or elements are necessary for comprehension. Both types of text can be fully understood even though some elements are skipped during reading. Typical examples are the articles “a” and “the” in English and closing brackets in Java. However, in Java a much larger portion of the text was omitted during reading or just perceived parafoveally. Programming languages prescribe a certain structure for the text, so many elements can be presumed to be present and therefore do not have to be foveated. This finding illustrates that natural-language text and source code are fundamentally different types of text.

Code reading is a perpetual programming activity, performed when learning to program, writing new code, during debugging, maintenance, and reviewing and plays an important role in program comprehension. The results of the EMCR study corroborate claims of its relevance by Busjahn & Schulte [2013], Deimel Jr. [1985], Kimura [1979], Lister et al. [2004], Pea [1986], Raymond [1991], Rooksby et al. [2006], Schulte [2007], Spinellis [2003b], and Xie et al. [2019], and they emphasize that code reading warrants more attention, both in research on software engineering and computer science education, and in programming instruction. It can reliably be concluded that reading source code is effectively different from reading natural-language text and that eye movements allow useful insights into different aspects of program comprehension. The robust methodology opens up the possibility to study comprehension with gaze data on a very detailed level and with more ecologically valid stimuli.

Central aspects of future work are refining the current methodology and broadening it to also include more data-driven components. Besides, the more general challenges of detecting oculomotor events and addressing errors in gaze data will continually be developed further with special consideration of program comprehension.

A large dataset about Eye Movements in Programming has recently been published, which is to a substantial extent based on the EMCR study presented here, allowing to apply the demonstrated measures to a rich body of data [Bednarik et al., 2020]. The EMIP dataset was collected in an international effort, it already contains samples of over 200 participants reading source code and is still growing. The participants’ level of programming expertise ranges from none to high and the stimuli consist of two programs, which are identical to the ones used for the second research question in the EMCR study. In addition to Java, the EMIP study contains the same stimulus programs in Python and Scala. Most EMIP participants read the programs in Java, but more samples in other languages are being added. The stimuli can also be translated to further programming languages without difficulty. Thus this dataset can eventually also be used to compare findings across programming languages and paradigms. Additionally to working with different programming languages, to determine the degree to which these results can be generalized, natural languages from further language families can also be included in order to ascertain how other reading approaches differ from code reading and how they affect it.

The EMCR analysis includes the measure AOI coverage to describe how much the gaze spread across the stimulus. In addition to this, it is highly worthwhile to inspect where the participants looked. With

regard to the stimuli it is of interest how much visual attention various code areas receive. As to level of expertise, it can be compared whether novices and experts focus on the same parts of a program. Such analyses can be carried out with AOIs on different levels of detail, e.g. assess which methods, lines, constructs, or even elements are looked at most by each group.

Moreover, the sequential structure of gaze during program comprehension can be analyzed in much greater detail. The proportion of linear reading operates on pairs of AOIs in the AOI sequence that were looked at consecutively, model occurrence and similarity on compressed AOI sequences of the complete trial. The intermediate step is to study sequences in between. On the one hand, only parts of the gaze sequence can be inspected, e.g. the first part in order to look for an initial scan of the program. On the other hand, instead of using an overall model sequence, it can be assessed how much participants exhibit smaller model behaviors, e.g. following a certain loop or other control structures. Again, different levels of detail are possible for AOIs, e.g. using block-AOIs to test transitions between methods. Further refinements for sequence-based analyses include exploring different scoring systems for alignments, e.g. using a matrix to assign specific scores to each pair of items. A major next step is to identify gaze patterns during program comprehension in a data-driven approach. The model behaviors Text Order and Execution Order were defined based on findings from literature and interviews with professional programmers. They were not derived from the EMCR gaze data. However, alignments and other computational techniques allow to search for common sub-sequences within the gaze, which ultimately yield visual strategies during program comprehension. Moreover, alignment scores can be used to establish whether the gaze of experts can be grouped into a cluster, in which the gaze sequences are more similar to each other than to the ones from novices. If such a group of expert data can be formed, it can be tested how similar a given gaze sequence is to the expert cluster.

The presented methodology is also suitable to study other programming-related tasks like debugging, other model behaviors, e.g. following the data-flow within a program, or differentiate between overall comprehension and particular processes like tracing. It also allows to advance the inclusion of readability aspects in programming, not just regarding the surface, e.g. finding an especially suitable formatting, but also conceptually when designing programming languages. Finally, in order to move forward from observable behavior to cognitive concepts, gaze data is to be complemented with qualitative data, e.g. from interviews with programmers of varying level of expertise as well as from retrospective think-aloud cued with eye movements.

Apart from the research perspective, findings from studying eye movements during program comprehension also have implications for programming education as well as for practicing programmers. The EMCR study demonstrated that reading source code calls for a less linear reading behavior than reading natural-languages like English. Advising programming novices early on that their customary reading behavior is not the most adequate approach for source code could save them some effort and frustration. Further work is needed in order to identify expert code reading strategies more comprehensively, and to develop as well as evaluate corresponding teaching approaches. Nevertheless, a lightweight intervention at least for teaching Java can be to instruct learners to purposefully head for the `main`-method very early on when trying to understand a program. Providing IDEs by default with the option to directly jump to the `main` or another predefined reference point can facilitate working with large programs, especially during maintenance and code review. Consequently, using eye movements to study how visual attention is allocated during different programming-related activities, like understanding a given program or debugging, leads to insights about the cognitive aspects involved in programming which can be used to improve programming instruction, but also to support software engineers in their work. There is still much to be learned about program comprehension and gaze proved to be a valuable vehicle for that.

Bibliography

Albert, W. & Tedesco, D. (2010). Reliability of self-reported awareness measures based on eye tracking. *Journal of Usability Studies*, 5(2), 50–64.

Anderson, N. C., Anderson, F., Kingstone, A., & Bischof, W. F. (2015). A comparison of scanpath comparison methods. *Behavior Research Methods*, 47(4), 1377–1392.

Aschwanden, C. & Crosby, M. (2006). Code scanning patterns in program comprehension. In *Proceedings of the 39th Hawaii International Conference on System Science*.

Ashmore, M., Duchowski, A. T., & Shoemaker, G. (2005). Efficient eye pointing with a fisheye lens. In *Proceedings of Graphics Interface 2005* (pp. 203–210).

Bates, R. & Istance, H. (2002). Zooming interfaces! Enhancing the performance of eye controlled pointing devices. In *Proceedings of the Fifth International ACM Conference on Assistive Technologies*, Assets '02 (pp. 119–126). New York, NY, USA: ACM.

Bednarik, R., Busjahn, T., Gibaldi, A., Ahadi, A., Bielikova, M., Crosby, M., Essig, K., Fagerholm, F., Jbara, A., Lister, R., Orlov, P., Paterson, J., Sharif, B., Sirkiä, T., Stelovsky, J., Tvarozek, J., Vrzakova, H., & Linde, I. v. d. (2020). EMIP: The eye movements in programming dataset. *Science of Computer Programming*, 198, 102520.

Bednarik, R., Busjahn, T., & Schulte, C., Eds. (2014). *Eye Movements in Programming Education: Analyzing the Expert's Gaze*. Number 18 in Reports and Studies in Forestry and Natural Sciences. Joensuu, Finland: University of Eastern Finland.

Bednarik, R., Busjahn, T., Schulte, C., & Tamm, S., Eds. (2016). *Eye Movements in Programming: Models to Data*. Number 23 in Reports and Studies in Forestry and Natural Sciences. Joensuu, Finland: University of Eastern Finland.

Bednarik, R. & Schulte, C., Eds. (2018). *EMIP '18: Proceedings of the Workshop on Eye Movements in Programming*, New York, NY, USA. ACM.

Bednarik, R. & Tukiainen, M. (2004a). Visual attention and representation switching in Java program debugging: A study using eye movement tracking. In *Proceedings of the 16th Workshop of the Psychology of Programming Interest Group* (pp. 159–169). Carlow, Ireland.

Bednarik, R. & Tukiainen, M. (2004b). Visual attention tracking during program debugging. In *Proceedings of the Third Nordic Conference on Human-Computer Interaction*, NordiCHI '04 (pp. 331–334). New York, NY, USA: ACM.

Bednarik, R. & Tukiainen, M. (2006). An eye-tracking methodology for characterizing program comprehension processes. In *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications*, ETRA '06 (pp. 125–132). New York, NY, USA: ACM.

Bednarik, R. & Tukiainen, M. (2007). Validating the Restricted Focus Viewer: A study using eye-movement tracking. *Behavior Research Methods*, 39(2), 274–282.

Beelders, T. & Plessis, J.-P. d. (2016a). The influence of syntax highlighting on scanning and reading behaviour for source code. In *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists*, SAICSIT '16 (pp. 1–10). New York, NY, USA: ACM.

Beelders, T. R. & Plessis, J.-P. L. d. (2016b). Syntax highlighting as an influencing factor when reading and comprehending source code. *Journal of Eye Movement Research*, 9(1), 1–11.

Begel, A. & Siegmund, J., Eds. (2019). *EMIP '19; Proceedings of the 6th International Workshop on Eye Movements in Programming*, Montreal, Quebec, Canada. IEEE Press.

Bente, G. (2004). Erfassung und Analyse des Blickverhaltens. In R. Mangold, P. Vorderer, & G. Bente (Eds.), *Lehrbuch der Medienpsychologie* (pp. 297–324). Göttingen, Bern, Toronto, Seattle: Hogrefe-Verlag.

Binkley, D., Davis, M., Lawrie, D., Maletic, J. I., Morrell, C., & Sharif, B. (2012). The impact of identifier style on effort and comprehension. *Empirical Software Engineering*, 18(2), 219–276.

Blackwell, A. F., Jansen, A. R., & Marriott, K. (2000). Restricted Focus Viewer: A tool for tracking visual attention. In M. Anderson, P. Cheng, & V. Haarslev (Eds.), *Theory and Application of Diagrams*, volume 1889 of *Lecture Notes in Computer Science* (pp. 162–177).: Springer Berlin Heidelberg.

Blascheck, T. & Sharif, B. (2019). Visually analyzing eye movements on natural language texts and source code snippets. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, ETRA '19 (pp. 1–9). New York, NY, USA: ACM.

Blignaut, P. (2009). Fixation identification: The optimum threshold for a dispersion algorithm. *Attention, Perception, & Psychophysics*, 71(4), 881–895.

Blignaut, P. & Beelders, T. (2009). The effect of fixational eye movements on fixation identification with a dispersion-based fixation detection algorithm. *Journal of Eye Movement Research*, 2(5).

Böckenhauer, H.-J. & Bongartz, D. (2003). *Algorithmische Grundlagen der Bioinformatik: Modelle, Methoden und Komplexität*. Leitfäden der Informatik. Stuttgart, Leipzig, Wiesbaden: Teubner.

Budde, L., Heinemann, B., & Schulte, C. (2017). A theory based tool set for analysing reading processes in the context of learning programming. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, WiPSCE '17 (pp. 83–86). New York, NY, USA: ACM.

Busjahn, T., Bednarik, R., Begel, A., Crosby, M., Paterson, J. H., Schulte, C., Sharif, B., & Tamm, S. (2015a). Eye movements in code reading: Relaxing the linear order. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*, ICPC '15 (pp. 255–265). USA: IEEE Computer Society.

Busjahn, T., Bednarik, R., & Schulte, C. (2014a). What influences dwell time during source code reading? Analysis of element type and frequency as factors. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '14 (pp. 335–338). New York, NY, USA: ACM.

Busjahn, T. & Schulte, C. (2013). The use of code reading in teaching programming. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, Koli Calling '13 (pp. 3–11). New York, NY, USA: ACM.

Busjahn, T., Schulte, C., & Busjahn, A. (2011). Analysis of code reading to gain more insight in program comprehension. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, Koli Calling '11 (pp. 1–9). New York, NY, USA: ACM.

Busjahn, T., Schulte, C., & Kropp, E. (2014b). Developing coding schemes for program comprehension using eye movements. In *Proceedings 25th Annual Workshop of the Psychology of Programming Interest Group* (pp. 111–122). Brighton, UK.

Busjahn, T., Schulte, C., Sharif, B., Simon, Begel, A., Hansen, M., Bednarik, R., Orlov, P., Ihantola, P., Shchekotova, G., & Antropova, M. (2014c). Eye tracking in computing education. In *Proceedings of the Tenth Annual Conference on International Computing Education Research*, ICER '14 (pp. 3–10). New York, NY, USA: ACM.

Busjahn, T., Schulte, C., Tamm, S., & Bednarik, R., Eds. (2015b). *Eye Movements in Programming Education II: Analyzing the Novice's Gaze.* Number TR-B-15-01 in Technical Reports Serie B. Berlin, Germany: Freie Universität Berlin, Department of Mathematics and Computer Science.

Campbell, W. & Bolker, E. (2002). Teaching programming by immersion, reading and writing. *32nd Annual Frontiers in Education*, 1, 23–28.

Carl, M. (2013). Dynamic programming for re-mapping noisy fixations in translation tasks. *Journal of Eye Movement Research*, 6(2), 1–11.

Cerrolaza, J. J., Villanueva, A., Villanueva, M., & Cabeza, R. (2012). Error characterization and compensation in eye tracking systems. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12 (pp. 205–208). New York, NY, USA: ACM.

Chapanis, A. (1951). Theory and methods for analyzing errors in man-machine systems. *Annals of the New York Academy of Sciences*, 51(7), 1179–1203.

Clear, T., Whalley, J. L., Robbins, P., Philpott, A., Eckerdal, A., & Laakso, M.-J. (2011). Report on the final BRACElet workshop: Auckland University of Technology, September 2010. *Journal of Applied Computing and Information Technology*, 15(1).

Cohen, A. L. (2012). Software for the automatic correction of recorded eye fixation locations in reading experiments. *Behavior Research Methods*, 45(3), 679–683.

Cristino, F., Mathôt, S., Theeuwes, J., & Gilchrist, I. D. (2010). ScanMatch: A novel method for comparing fixation sequences. *Behavior Research Methods*, 42(3), 692–700.

Crosby, M. & Stelovsky, J. (1989). The influence of user experience and presentation medium on strategies of viewing algorithms. In *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences, 1989. Vol. II: Software Track*, volume 2 (pp. 438–446). Los Alamitos, CA, USA: IEEE Computer Society.

Crosby, M. E. (1986). *Natural versus computer languages: A reading comparison.* PhD thesis, University of Hawai'i.

Crosby, M. E., Scholtz, J., & Wiedenbeck, S. (2002). The roles beacons play in comprehension for novice and expert programmers. In J. Kuljis, B. L., & R. Scoble (Eds.), *Proceedings of the 14th Workshop of the Psychology of Programming Interest Group* (pp. 58–73).

Crosby, M. E. & Stelovsky, J. (1990). How do we read algorithms? A case study. *Computer*, 23(1), 24–35.

Daw, N. (2012). *How Vision Works: The Physiological Mechanisms Behind What We See.* New York: Oxford University Press.

Day, R.-F. (2010). Examining the validity of the Needleman-Wunsch algorithm in identifying decision strategy with eye-movement data. *Decision Support Systems*, 49(4), 396–403.

Deimel, L. E. & Naveda, J. F. (1990). *Reading Computer Programs: Instructor's Guide and Exercises.* Technical report, Carnegie-Mellon University.

Deimel Jr., L. E. (1985). The uses of program reading. *SIGCSE Bull.*, 17(2), 5–14.

Dolezalova, J. & Popelka, S. (2016). ScanGraph: A novel scanpath comparison method using visualisation of graph cliques. *Journal of Eye Movement Research*, 9(4).

Drewes, J., Masson, G. S., & Montagnini, A. (2012). Shifts in reported gaze position due to changes in pupil size: ground truth and compensation. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12 (pp. 209–212). New York, NY, USA: ACM.

Dubochet, G. (2009). Computer code as a medium for human communication: Are programming languages improving? In *Proceedings of the 21st Working Conference on the Psychology of Programmers Interest Group* (pp. 174–187). Limerick, Ireland.

Duchowski, A. T. (2017). *Eye Tracking Methodology: Theory and Practice*. Springer, 3 edition.

Ethnologue (2020). What is the most spoken language? https://www.ethnologue.com/guides/most-spoken-languages. [last accessed 12/05/2020].

Eysel, U. (2019). Sehen. In R. Brandes, F. Lang, & R. F. Schmidt (Eds.), *Physiologie des Menschen: mit Pathophysiologie*, Springer-Lehrbuch (pp. 721–769). Berlin, Heidelberg: Springer, 32. edition.

Fan, Q. (2010). *The Effects of Beacons, Comments, and Tasks on Program Comprehension Process in Software Maintenance*. PhD thesis, University of Maryland at Baltimore County, Catonsville, MD, USA.

Feit, A. M., Williams, S., Toledo, A., Paradiso, A., Kulkarni, H., Kane, S., & Morris, M. R. (2017). Toward everyday gaze input: Accuracy and precision of eye tracking and implications for design. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17 (pp. 1118–1130). New York, NY, USA: ACM.

Frank, M. C., Vul, E., & Saxe, R. (2012). Measuring the development of social attention using free-viewing. *Infancy*, 17(4), 355–375.

Friedman, L., Rigas, I., Abdulin, E., & Komogortsev, O. V. (2018). A novel evaluation of two related and two independent algorithms for eye movement classification during reading. *Behavior Research Methods*, 50(4), 1374–1397.

Fritz, T., Begel, A., Müller, S. C., Yigit-Elliott, S., & Züger, M. (2014). Using psycho-physiological measures to assess task difficulty in software development. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014 (pp. 402–413). New York, NY, USA: ACM.

Gagl, B., Hawelka, S., & Hutzler, F. (2011). Systematic influence of gaze position on pupil size measurement: analysis and correction. *Behavior Research Methods*, 43(4), 1171–1181.

Gog, T. v., Kester, L., Nievelstein, F., Giesbers, B., & Paas, F. (2009). Uncovering cognitive processes: Different techniques that can contribute to cognitive load research and instruction. *Computers in Human Behavior*, 25(2), 325–331.

Guo, P. (2014). Python is now the most popular introductory teaching language at top - U.S. universities. https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext. [last accessed 12/05/2020].

Guéhéneuc, Y.-G. (2006). TAUPE: Towards understanding program comprehension. In *Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '06 (pp. 1–13). USA: IBM Corp.

Hansen, M. (2015). *Quantifying code complexity and comprehension*. PhD thesis, Indiana University, Bloomington, IN.

Haque, W., Aravind, A., & Reddy, B. (2009). Pairwise sequence alignment algorithms: A survey. In *Proceedings of the 2009 Conference on Information Science, Technology and Applications*, ISTA '09 (pp. 96–103). New York, NY, USA: ACM.

Hilburn, T. B., Towhidnejad, M., & Salamah, S. (2011). Read before you write. In *Proceedings of the 2011 24th IEEE-CS Conference on Software Engineering Education and Training*, CSEET '11 (pp. 371–380). USA: IEEE Computer Society.

Holmqvist, K., Nyström, M., Andersson, R., Dewhurst, R., Jarodzka, H., & Weijer, J. v. d. (2011). *Eye tracking: A comprehensive guide to methods and measures.* OUP Oxford.

Holmqvist, K., Nyström, M., & Mulvey, F. (2012). Eye tracker data quality: What it is and how to measure it. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12 (pp. 45–52). New York, NY, USA: ACM.

Hornof, A. J. & Halverson, T. (2002). Cleaning up systematic error in eye-tracking data by using required fixation locations. *Behavior Research Methods, Instruments, & Computers*, 34(4), 592–604.

Hyrskykari, A. (2006). Utilizing eye movements: Overcoming inaccuracy while tracking the focus of attention during reading. *Computers in Human Behavior*, 22(4), 657–671.

Inhoff, A. W. & Radach, R. (1998). Definition and computation of oculomotor measures in the study of cognitive processes. In G. Underwood (Ed.), *Eye Guidance in Reading and Scene Perception* (pp. 29–53). Amsterdam: Elsevier Science Ltd.

Jansen, A. R., Blackwell, A. F., & Marriott, K. (2003). A tool for tracking visual attention: The Restricted Focus Viewer. *Behavior Research Methods, Instruments, & Computers*, 35(1), 57–69.

John, S., Weitnauer, E., & Koesling, H. (2012). Entropy-based correction of eye tracking data for static scenes. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12 (pp. 297–300). New York, NY, USA: ACM.

Just, M. A. & Carpenter, P. A. (1980). A theory of reading: From eye fixations to comprehension. *Psychological Review*, 87(4), 329–354.

Karn, K. S. (2000). "Saccade pickers" vs. "fixation pickers": The effect of eye tracking instrumentation on research. In *Proceedings of the 2000 Symposium on Eye Tracking Research & Applications*, ETRA '00 (pp. 87–88). New York, NY, USA: ACM.

Karsh, R. & Breitenbach, F. W. (1983). Looking at looking: The amorphous fixation measure. In *Eye movements and psychological functions: International views* (pp. 53–64). Hillsdale (NJ), London: Lawrence Erlbaum Associates.

Kerkau, F. (2011). Usability-Testing zur Qualitätssicherung von Online-Lernangeboten. In P. Klimsa & L. J. Issing (Eds.), *Online-Lernen: Handbuch für Wissenschaft und Praxis* (pp. 329–337). München: Oldenbourg Verlag, 2. edition.

Kimura, T. (1979). Reading before composition. *SIGCSE Bull.*, 11(1), 162–166.

Kintsch, W. (1998). *Comprehension: A paradigm for cognition.* Cambridge University Press.

Kölling, M. & Rosenberg, J. (2001). Guidelines for teaching object orientation with Java. *SIGCSE Bull.*, 33(3), 33–36.

Komogortsev, O. V., Gobert, D. V., Jayarathna, S., Koh, D. H., & Gowda, S. M. (2010). Standardization of automated analyses of oculomotor fixation and saccadic behaviors. *IEEE Transactions on Biomedical Engineering*, 57(11), 2635–2645.

Komogortsev, O. V., Jayarathna, S., Koh, D. H., & Gowda, S. M. (2009). *Qualitative and Quantitative Scoring and Evaluation of the Eye Movement Classification Algorithms.* Technical Report TXSTATE-CS-TR-2009-16, Department of Computer Science, San Marcos, TX.

Korda, A. I., Asvestas, P. A., Matsopoulos, G. K., Ventouras, E. M., & Smyrnis, N. (2018). Automatic identification of eye movements using the largest lyapunov exponent. *Biomedical Signal Processing and Control*, 41, 10–20.

Kruskal, J. B. (1983). An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM Review*, 25(2), 201–237.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10 (pp. 707–710).

Lin, Y., Wu, C., Hou, T., Lin, Y., Yang, F., & Chang, C. (2016). Tracking students' cognitive processes during program debugging — An eye-movement approach. *IEEE Transactions on Education*, 59(3), 175–186.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *SIGCSE Bull.*, 36(4), 119–150.

Lister, R., Fidge, C., & Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *SIGCSE Bull.*, 41(3), 161–165.

Liversedge, S. P., Gilchrist, I. D., & Everling, S., Eds. (2011). *The Oxford Handbook of Eye Movements*. Oxford: Oxford University Press.

Lohmeier, S. (2015). Experimental evaluation and modelling of the comprehension of indirect anaphors in a programming language. http://www.monochromata.de/master_thesis/ma1.3.pdf. [last accessed 12/05/2020].

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. In *Proceeding of the Fourth International Workshop on Computing Education Research*, ICER '08 (pp. 101–112). New York, NY, USA: ACM.

MacKenzie, I. S. & Zhang, X. (2008). Eye typing using word and letter prediction and a fixation algorithm. In *Proceedings of the 2008 Symposium on Eye Tracking Research & Applications*, ETRA '08 (pp. 55–58). New York, NY, USA: ACM.

Majaranta, P. & Bulling, A. (2014). Eye tracking and eye-based human–computer interaction. In S. H. Fairclough & K. Gilleade (Eds.), *Advances in Physiological Computing*, Human–Computer Interaction Series (pp. 39–65). London: Springer.

Mannila, L. (2007). Novices' progress in introductory programming courses. *Informatics in Education*, 6(1), 139–152.

Martínez-Gómez, P. & Aizawa, A. (2014). Recognition of understanding level and language skill using measurements of reading behavior. In *Proceedings of the 19th International Conference on Intelligent User Interfaces*, IUI '14 (pp. 95–104). New York, NY, USA: ACM.

Martinez-Gomez, P., Chen, C., Hara, T., Kano, Y., & Aizawa, A. (2012). Image registration for text-gaze alignment. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces*, IUI '12 (pp. 257–260). New York, NY, USA: ACM.

Mason, R. & Cooper, G. (2014). Introductory programming courses in Australia and New Zealand in 2013 - trends and reasons. In *Proceedings of the Sixteenth Australasian Computing Education Conference*, ACE '14 (pp. 139–147). Auckland, New Zealand: Australian Computer Society, Inc.

Mason, R., Cooper, G., & de Raadt, M. (2012). Trends in introductory programming courses in Australian universities - Languages, environments and pedagogy. In *Proceedings of the Fourteenth Australasian Computing Education Conference*, ACE '12 (pp. 33–42). Melbourne, Australia: Australian Computer Society, Inc.

Mathôt, S., Cristino, F., Gilchrist, I. D., & Theeuwes, J. (2012). A simple way to estimate similarity between pairs of eye movement sequences. *Journal of Eye Movement Research*, 5(1), 1–15.

Mayrhauser, A. v. & Vans, A. M. (1994). *Program Unterstanding: A Survey*. Number CS-94-120. Colorado State University.

Mazzei, A., Eivazi, S., Marko, Y., Kaplan, F., & Dillenbourg, P. (2014). 3D model-based gaze estimation in natural reading: a systematic error correction procedure based on annotated texts. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '14 (pp. 87–90). New York, NY, USA: ACM.

Mele, M. L. & Federici, S. (2012). Gaze and eye-tracking solutions for psychological research. *Cognitive Processing*, 13(1), 261–265.

Merrienboer, J. J. G. & Krammer, H. P. M. (1987). Instructional strategies and tactics for the design of introductory computer programming courses in high school. *Instructional Science*, 16(3), 251–285.

Miniotas, D., Špakov, O., & MacKenzie, I. S. (2004). Eye gaze interaction with expanding targets. In *CHI '04 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '04 (pp. 1255–1258). New York, NY, USA: ACM.

Mishra, A., Carl, M., & Bhattacharya, P. (2012). A heuristic-based approach for systematic error correction of gaze data for reading. In *Proceedings of the First Workshop on Eye-tracking and Natural Language Processing* (pp. 71–80).

Moreno, A., Myller, N., Sutinen, E., & Ben-Ari, M. (2004). Visualizing programs with Jeliot 3. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04 (pp. 373–376). New York, NY, USA: ACM.

Murphy, E., Crick, T., & Davenport, J. H. (2017). An analysis of introductory programming courses at UK Universities. *The Art, Science, and Engineering of Programming*, 1(2).

Needleman, S. B. & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3), 443–453.

Nelson, G. L., Xie, B., & Ko, A. J. (2017). Comprehension First: Evaluating a novel pedagogy and tutoring system for program tracing in CS1. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER '17 (pp. 2–11). New York, NY, USA: ACM.

Nevalainen, S. & Sajaniemi, J. (2004). Comparison of three eye tracking devices in psychology of programming research. In *Proceedings of 16th Annual Workshop of the Psychology of Programming Interest Group (PPIG'04)* (pp. 151–158).

Niehorster, D. C., Cornelissen, T. H. W., Holmqvist, K., Hooge, I. T. C., & Hessels, R. S. (2018). What to expect from your remote eye-tracker when participants are unrestrained. *Behavior Research Methods*, 50(1), 213–227.

Nüssli, M.-A. (2011). *Dual Eye-Tracking Methods for the Study of Remote Collaborative Problem Solving.* PhD thesis, École Polytechnique Fédérale de Lausanne, Lausanne.

Nyström, M., Andersson, R., Holmqvist, K., & Weijer, J. v. d. (2012). The influence of calibration method and eye physiology on eyetracking data quality. *Behavior Research Methods*, 45(1), 272–288.

Nyström, M. & Holmqvist, K. (2010). An adaptive algorithm for fixation, saccade, and glissade detection in eyetracking data. *Behavior Research Methods*, 42(1), 188–204.

Obaidellah, U., Haek, M. A., & Cheng, P. C.-H. (2018). A survey on the usage of eye-tracking in computer programming. *ACM Comput. Surv.*, 51(1), 1–58.

Olsen, A. & Matos, R. (2012). Identifying parameter values for an I-VT fixation filter suitable for handling data sampled with various sampling frequencies. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12 (pp. 317–320). New York, NY, USA: ACM.

Orlov, P. A. (2016). *Extrafoveal Vision During Source Code Comprehension: a Gaze-contingent Tool, a Latency Evaluation Method, and Experiments.* PhD thesis, University of Eastern Finland, Joensuu, Finland.

Palmer, C. & Sharif, B. (2016). Towards automating fixation correction for source code. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*, ETRA '16 (pp. 65–68). New York, NY, USA: ACM.

Pazos, F. & Chagoyen, M. (2015). Sequences. In *Practical Protein Bioinformatics* (pp. 1–41). Springer International Publishing.

Pea, R. D. (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, 2(1), 25–36.

Peachock, P., Iovino, N., & Sharif, B. (2017). Investigating eye movements in natural language and C++ source code - A replication experiment. In D. D. Schmorrow & C. M. Fidopiastis (Eds.), *Augmented Cognition. Neurocognition and Machine Learning*, volume 10284 of *Lecture Notes in Computer Science* (pp. 206–218).: Springer International Publishing.

Pennington, N. (1987). Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, 19(3), 295–341.

Perkins, D. N. & Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers. In *Papers presented at the First Workshop on Empirical Studies of Programmers* (pp. 213–229). Washington, D.C., United States: Ablex Publishing Corp.

Peterson, C. S., Abid, N. J., Bryant, C. A., Maletic, J. I., & Sharif, B. (2019). Factors influencing dwell time during source code reading: A large-scale replication experiment. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, ETRA '19 (pp. 1–4). New York, NY, USA: ACM.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Raymond, D. R. (1991). Reading source code. In *Proceedings of the 1991 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '91 (pp. 3–16).: IBM Press.

Rayner, K. (1998). Eye movements in reading and information processing: 20 years of research. *Psychological Bulletin*, 124(3), 372–422.

Rayner, K., Juhasz, B. J., & Pollatsek, A. (2005). Eye movements during reading. In M. J. Snowling & C. Hulme (Eds.), *The Science of Reading: A Handbook* (pp. 79–97). John Wiley & Sons, Ltd.

Rayner, K., Pollatsek, A., Ashby, J., & Clifton Jr., C. (2012). *Psychology of Reading*. New York, London: Psychology Press, 2. edition.

Recurity Labs (2020). Recurity Labs Website. https://www.recurity-labs.com/trainings/code. [last accessed 12/05/2020].

Romero, P., Boulay, B. d., Lutz, R., & Cox, R. (2003). The effects of graphical and textual visualisations in multi-representational debugging environments. In *IEEE Symposium on Human Centric Computing Languages and Environments, 2003* (pp. 236–238).

Romero, P., Cox, R., Boulay, B. d., & Lutz, R. (2002a). Visual attention and representation switching during Java program debugging: A study using the Restricted Focus Viewer. In M. Hegarty, B. Meyer, & N. H. Narayanan (Eds.), *Diagrammatic Representation and Inference*, Lecture Notes in Computer Science (pp. 221–235). Berlin, Heidelberg: Springer.

Romero, P., Lutz, R., Cox, R., & Boulay, B. d. (2002b). Co-ordination of multiple external representations during Java program debugging. In *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments* (pp. 207–214).

Rooksby, J., Martin, D., & Rouncefield, M. (2006). Reading as part of computer programming. An ethnomethodological enquiry. In *Proceedings of the 18th Workshop of the Psychology of Programming Interest Group* (pp. 198–212). University of Sussex.

Salvucci, D. D. & Goldberg, J. H. (2000). Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 Symposium on Eye Tracking Research & Applications*, ETRA '00 (pp. 71–78). New York, NY, USA: ACM.

Schall, A. & Romano Bergstrom, J. (2014). Introduction to eye tracking. In J. Romano Bergstrom & A. J. Schall (Eds.), *Eye Tracking in User Experience Design* (pp. 3–26). Boston: Morgan Kaufmann.

Schulte, C. (2007). Lesen im Informatikunterricht. In S. Schubert (Ed.), *Didaktik der Informatik in Theorie und Praxis: 12. GI-Fachtagung Informatik und Schule - INFOS 2007, Siegen* (pp. 307–318). Bonn: Köllen Verlag.

Schulte, C. (2008a). Block Model - an educational model of program comprehension as a tool for a scholarly approach to teaching. In *Proceedings of the Fourth International Workshop on Computing Education Research*, ICER '08 (pp. 149–160). New York, NY, USA: ACM.

Schulte, C. (2008b). Duality reconstruction – Teaching digital artifacts from a socio-technical perspective. In R. T. Mittermeir & M. M. Sysło (Eds.), *Informatics Education - Supporting Computational Thinking*, Lecture Notes in Computer Science (pp. 110–121). Berlin, Heidelberg: Springer.

Schulte, C., Clear, T., Taherkhani, A., Busjahn, T., & Paterson, J. H. (2010). An introduction to program comprehension for computer science educators. In *Proceedings of the 2010 ITiCSE Working Group Reports*, ITiCSE-WGR '10 (pp. 65–86). New York, NY, USA: ACM.

Selby, C. (2011). Four approaches to teaching programming. In *Learning, Media and Technology: a doctoral research conference*. London.

SensoMotoric Instruments (2014). iView X^{TM} SDK: v3.6.

SensoMotoric Instruments (2016a). RED-m Spec sheet.

SensoMotoric Instruments (2016b). RED250mobile Spec sheet.

Sharafi, Z., Shaffer, T., Sharif, B., & Guéhéneuc, Y.-G. (2015a). Eye-tracking metrics in software engineering. In *2015 Asia-Pacific Software Engineering Conference (APSEC)* (pp. 96–103).

Sharafi, Z., Sharif, B., Guéhéneuc, Y.-G., Begel, A., Bednarik, R., & Crosby, M. (2020). A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering*, 25(5), 3128–3174.

Sharafi, Z., Soh, Z., & Guéhéneuc, Y.-G. (2015b). A systematic literature review on the usage of eye-tracking in software engineering. *Information and Software Technology*, 67, 79–107.

Sharif, B., Falcone, M., & Maletic, J. I. (2012). An eye-tracking study on the role of scan time in finding source code defects. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12 (pp. 381–384). New York, NY, USA: ACM.

Sharif, B., Jetty, G., Aponte, J., & Parra, E. (2013). An empirical study assessing the effect of SeeIT 3D on comprehension. *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*, (pp. 1–10).

Sharif, B. & Shaffer, T. (2015). The use of eye tracking in software development. In D. D. Schmorrow & C. M. Fidopiastis (Eds.), *Foundations of Augmented Cognition*, volume 9183 of *Lecture Notes in Computer Science* (pp. 807–816). Springer International Publishing.

Shic, F., Scassellati, B., & Chawarska, K. (2008). The incomplete fixation measure. In *Proceedings of the 2008 Symposium on Eye Tracking Research & Applications*, ETRA '08 (pp. 111–114). New York, NY, USA: ACM.

Siegmund, J., Begel, A., & Peitek, N. (2019). Summary of the Sixth Edition of the International Workshop on Eye Movements in Programming. *SIGSOFT Softw. Eng. Notes*, 44(3), 54–55.

Simon (2014). Eye movements in programming education: analysing the expert's gaze. In R. Bednarik, T. Busjahn, & C. Schulte (Eds.), *Eye Movements in Programming Education: Analyzing the Expert's Gaze* (pp. 27–29). Joensuu, Finland: University of Eastern Finland.

Simon (2015). Eye movements in programming education 2: Analysing the novice's gaze. In T. Busjahn, C. Schulte, S. Tamm, & R. Bednarik (Eds.), *Eye Movements in Programming Education II: Analyzing the Novice's Gaze*, volume TR-B-15-01 of *Freie Universität Berlin, Fachbereich Mathematik und Informatik*. Berlin, Germany: Freie Universität Berlin.

Skovsgaard, H., Mateo, J. C., Flach, J. M., & Hansen, J. P. (2010). Small-target selection with gaze alone. In *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications*, ETRA '10 (pp. 145–148). New York, NY, USA: ACM.

Smith, T. F. & Waterman, M. S. (1981). Comparison of biosequences. *Advances in Applied Mathematics*, 2(4), 482–489.

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. *Commun. ACM*, 29(9), 850–858.

Soloway, E. & Ehrlich, K. (1984). Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, 10(5), 595–609.

Špakov, O. & Gizatdinova, Y. (2014). Real-time hidden gaze point correction. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '14 (pp. 291–294). New York, NY, USA: ACM.

Špakov, O., Istance, H., Hyrskykari, A., Siirtola, H., & Räihä, K.-J. (2018). Improving the performance of eye trackers with limited spatial accuracy and low sampling rates for reading analysis by heuristic fixation-to-word mapping. *Behavior Research Methods*, 51, 1–27.

Spinelli, L., Pandey, M., & Oney, S. (2018). Attention patterns for code animations: Using eye trackers to evaluate dynamic code presentation techniques. In *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming*, Programming'18 Companion (pp. 99–104). New York, NY, USA: ACM.

Spinellis, D. (2003a). *Code Reading: The Open Source Perspective*. Effective Software Development Series. Boston, San Francisco: Addison-Wesley.

Spinellis, D. (2003b). Reading, writing, and code. *Queue*, 1(7), 84–89.

Storey, M.-A. (2006). Theories, tools and research methods in program comprehension: past, present and future. *Software Quality Journal*, 14(3), 187–208.

Tamm, S., Bednarik, R., Busjahn, T., Schulte, C., Vrzakova, H., & Budde, L., Eds. (2017). *Eye Movements in Programming: Spring Academy 2017*. Number TR-B-17-02 in Technical Reports Serie B. Berlin, Germany: Freie Universität Berlin, Department of Mathematics and Computer Science.

TechWell Contributor (2001). What's so great about inspections?!? https://www.stickyminds.com/article/whats-so-great-about-inspections. [last accessed 12/05/2020].

Topić, G., Yamaya, A., Aizawa, A., & Martínez-Gómez, P. (2016). FixFix: Fixing the fixations. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*, ETRA '16 (pp. 319–320). New York, NY, USA: ACM.

Uwano, H., Nakamura, M., Monden, A., & Matsumoto, K.-i. (2006). Analyzing individual performance of source code review using reviewers' eye movement. In *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications*, ETRA '06 (pp. 133–140). New York, NY, USA: ACM.

Uwano, H., Nakamura, M., Monden, A., & Matsumoto, K.-i. (2007). Exploiting eye movements for evaluating reviewer's performance in software review. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E90-A(10), 2290–2300.

Velichkovsky, B., Sprenger, A., & Unema, P. (1997). Towards gaze-mediated interaction: Collecting solutions of the "Midas touch problem". In S. Howard, J. Hammond, & G. Lindgaard (Eds.), *Human-Computer Interaction INTERACT'97: IFIP TC13 International Conference on Human-Computer Interaction, 14th–18th July 1997, Sydney, Australia*, IFIP — The International Federation for Information Processing (pp. 509–516). Boston, MA: Springer US.

Velichkovsky, B. M., Dornhoefer, S. M., Pannasch, S., & Unema, P. J. (2000). Visual fixations and level of attentional processing. In *Proceedings of the 2000 Symposium on Eye Tracking Research & Applications*, ETRA '00 (pp. 79–85). New York, NY, USA: ACM.

Venables, A., Tan, G., & Lister, R. (2009). A closer look at tracing, explaining and code writing skills in the novice programmer. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop*, ICER '09 (pp. 117–128). New York, NY, USA: ACM.

Vidal, M., Pfeuffer, K., Bulling, A., & Gellersen, H. (2013). Pursuits: Eye-based interaction with moving targets. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13 (pp. 3147–3150). New York, NY, USA: ACM.

Voßkühler, A., Nordmeier, V., Kuchinke, L., & Jacobs, A. M. (2008). OGAMA (Open Gaze and Mouse Analyzer): Open-source software designed to analyze eye and mouse movements in slideshow study designs. *Behavior Research Methods*, 40(4), 1150–1162.

Wade, N. J. & Tatler, B. W. (2005). *The Moving Tablet of the Eye: The origins of modern eye movement research*. Oxford University Press.

West, J. M., Haake, A. R., Rozanski, E. P., & Karn, K. S. (2006). eyePatterns: Software for identifying patterns and similarities across fixation sequences. In *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications*, ETRA '06 (pp. 149–154). New York, NY, USA: ACM.

Widdel, H. (1984). Operational problems in analysing eye movements. In A. G. Gale & F. Johnson (Eds.), *Advances in Psychology*, volume 22 of *Theoretical and Applied Aspects of Eye Movement Research* (pp. 21–29). North-Holland.

Willoughby, C. E., Ponzin, D., Ferrari, S., Lobo, A., Landau, K., & Omidi, Y. (2010). Anatomy and physiology of the human eye: effects of mucopolysaccharidoses disease on structure and function – a review. *Clinical & Experimental Ophthalmology*, 38(s1), 2–11.

WorldAtlas (2019). What is the most spoken language in the world? https://www.worldatlas.com/articles/most-popular-languages-in-the-world.html. [last accessed 12/05/2020].

Wyatt, H. J. (2010). The human pupil and the use of video-based eyetrackers. *Vision Research*, 50(19), 1982–1988.

Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H., Tan, A. H., Hwa, L., Li, M., & Ko, A. J. (2019). A theory of instruction for introductory programming skills. *Computer Science Education*, 29(2-3), 205–253.

Xie, B., Nelson, G. L., & Ko, A. J. (2018). An explicit strategy to scaffold novice program tracing. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18 (pp. 344–349). New York, NY, USA: ACM.

Yamaya, A., Topić, G., & Aizawa, A. (2016). Fixation-to-Word mapping with classification of saccades. In *Companion Publication of the 21st International Conference on Intelligent User Interfaces*, IUI '16 Companion (pp. 36–40). New York, NY, USA: ACM.

Yamaya, A., Topić, G., & Aizawa, A. (2017). Vertical error correction using classification of transitions between sequential reading segments. *Journal of Information Processing*, 25, 100–106.

Yamaya, A., Topić, G., Martínez-Gómez, P., & Aizawa, A. (2015). Dynamic-programming–based method for fixation-to-word mapping. In R. Neves-Silva, L. C. Jain, & R. J. Howlett (Eds.), *Intelligent Decision Technologies*, volume 39 of *Smart Innovation, Systems and Technologies* (pp. 649–659). Springer International Publishing.

Zemblys, R., Niehorster, D. C., Komogortsev, O., & Holmqvist, K. (2018). Using machine learning to detect events in eye-tracking data. *Behavior Research Methods*, 50(1), 160–181.

Zhang, X., Ren, X., & Zha, H. (2008). Improving eye cursor's stability for eye pointing tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08 (pp. 525–534). New York, NY, USA: ACM.

Zhang, Y. & Hornof, A. J. (2011). Mode-of-disparities error correction of eye-tracking data. *Behavior Research Methods*, 43(3), 834–842.

Zhang, Y. & Hornof, A. J. (2014). Easy post-hoc spatial recalibration of eye tracking data. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '14 (pp. 95–98). New York, NY, USA: ACM.

Zimmermann, K.-H. (2003). Pairwise sequence alignment. In *An Introduction to Protein Informatics*, volume 749 of *The Kluwer International Series in Engineering and Computer Science* (pp. 47–74). Springer US.

List of figures

List of tables

List of abbreviations

AOI	Area of interest
EMIP	Eye movements in programming
EMCR	Eye movements in code reading
IDE	Integrated development environment
I-DT	Identification by dispersion threshold algorithm
NT	Natural-language text
OGAMA	OpenGazeAndMouseAnalyzer
PFL	Probable fixation location
POR	Point of regard
RFL	Required fixation location
RFV	Restricted focus viewer
RL	Reference location
RQ	Research question
SC	Source code
SD	Standard deviation

A

Appendix

A.1 Questionnaires

A.1.1 Novices

Questionnaire "Program comprehension"

Subject code: _____
It consists of the first letter of your mother's first name, the second letter of your place of birth and the first two numbers of your birthday (e.g. your mother's name: Anna, your place of birth: Berlin, your birthday: 26.03.1988 then the code is Ae26)

Age: _____

Gender: ☐ male ☐ female

Mother tongue: _____

Your English level: ☐ low ☐ medium ☐ high

Your overall programming expertise: ☐ none ☐ low ☐ medium ☐ high

Your expertise in Java: ☐ none ☐ low ☐ medium ☐ high

How long have you been programming: _____ years

How often do you use any programming language other than Java:
- ☐ less than 1 hour / month
- ☐ less than 1 hour / week
- ☐ less than 1 hour / day
- ☐ more than 1 hour / day

How many years experience do you have programming Java: _____ years

How often do you program in Java:
- ☐ less than 1 hour / month
- ☐ less than 1 hour / week
- ☐ less than 1 hour / day
- ☐ more than 1 hour / day

Which programming languages do you know:

Language:	Level of expertise:
_____	☐ low ☐ medium ☐ high
_____	☐ low ☐ medium ☐ high
_____	☐ low ☐ medium ☐ high
_____	☐ low ☐ medium ☐ high
_____	☐ low ☐ medium ☐ high

Do you have vision problems (myopia, astigmatism, ...)?

☐ no

☐ yes (if yes, please note which type of problems) _____

Are you currently wearing glasses or contact lenses?

☐ no

☐ yes, glasses

☐ yes, contact lenses

Are you currently wearing mascara or other eye-makeup?

☐ no

☐ yes

Your signature below means that you voluntarily agree to participate in this research study.

_____

Date, signature

A.1.2 Experts

Questionnaire "Program comprehension"

Subject code: _____
It consists of the first letter of your mother's first name, the second letter of your place of birth and the first two numbers of your birthday (e.g. your mother's name: Anna, your place of birth: Berlin, your birthday: 26.03.1988 then the code is Ae26)

Age: _____

Gender: ☐ male ☐ female

Mother tongue: _____

Your English level: ☐ low ☐ medium ☐ high

Your overall programming expertise: ☐ none ☐ low ☐ medium ☐ high

Your expertise in Java: ☐ none ☐ low ☐ medium ☐ high

How long have you been programming: _____ years

How often do you use any programming language other than Java:
- ☐ less than 1 hour / month
- ☐ less than 1 hour / week
- ☐ less than 1 hour / day
- ☐ more than 1 hour / day

How much of that time is spend in development (new code) and in maintenance (existing code)?
development: _____ % maintenance: _____ %

How many years experience do you have programming Java: _____ years

How often do you program in Java:
- ☐ less than 1 hour / month
- ☐ less than 1 hour / week
- ☐ less than 1 hour / day
- ☐ more than 1 hour / day

How much of that time is spend in development (new code) and in maintenance (existing code)?
development: _____ % maintenance: _____ %

How large was your biggest Java project:
- ☐ 1 class
- ☐ 5-10 classes
- ☐ 10-50 classes
- ☐ More than 50 classes

Which programming languages do you know:

Language:	Level of expertise:
_____	☐ low ☐ medium ☐ high
_____	☐ low ☐ medium ☐ high
_____	☐ low ☐ medium ☐ high
_____	☐ low ☐ medium ☐ high
_____	☐ low ☐ medium ☐ high

How many years have you been a professional programmer: _____ years

Do you have vision problems (myopia, astigmatism, ...)?
- ☐ no
- ☐ yes (if yes, please note which type of problems) _____

Are you currently wearing glasses or contact lenses?

☐ no
☐ yes, glasses
☐ yes, contact lenses

Are you currently wearing mascara or other eye-makeup?

☐ no
☐ yes

Your signature below means that you voluntarily agree to participate in this research study.

_____

Date, signature

A.2 Natural-language stimuli

Instructions

Multiple-choice:
Please read and comprehend the following text. When you are done, press the left mouse button. Then you will be given a MULTIPLE-CHOICE question about the content.

Outcome:
Please read and comprehend the following text. When you are done, press the left mouse button. Then you will be asked about the text's OUTCOME.

Summary:
Please read and comprehend the following text. When you are done, press the left mouse button. Then you will be asked to give a SUMMARY.

A.2.1 NT1

> The invention of rockets is linked inextricably with the invention of 'black powder'. Most historians of technology credit the Chinese with its discovery. They base their belief on studies of Chinese writings or on the notebooks of early Europeans who settled in or made long visits to China to study its history and civilisation. It is probable that, some time in the tenth century, black powder was first compounded from its basic ingredients of saltpetre, charcoal and sulphur.

Comprehension questions:

Multiple-choice:
Which statement describes the content of the text?

- Black powder was probably discovered some time in the tenth century by the Chinese. It is essential for the invention of rockets.
- Historians believe that black powder was discovered by the Europeans after visits to China.
- Based on Chinese and European accounts, rockets were essential for the invention of black powder.
- I'm not sure.

Outcome:
Where was black powder supposedly discovered?
(If you don't know the answer, type '?'.)

Summary:
Please give a summary of the text.

A.2.2 NT2

Introducing dung beetles into a pasture is a simple process: approximately 1,500 beetles are released, a handful at a time, into fresh cow pats in the cow pasture. The beetles immediately disappear beneath the pats digging and tunnelling and, if they successfully adapt to their new environment, soon become a permanent, self-sustaining part of the local ecology. In time they multiply and within three or four years the benefits to the pasture are obvious.

Comprehension questions:

Multiple-choice:
Which statement describes the content of the text?

- Dung beetles dig themselves into pastures and have permanently to be maintained.
- Dung beetles live on cow pats and have permanently to be maintained.
- Dung beetles dig themselves into pastures and become a self-sustaining part of the ecosystem.
- I'm not sure.

Outcome:
Can pastures benefit from dung beetles?
(If you don't know the answer, type '?'.)

Summary:
Please give a summary of the text.

A.2.3 NT3

The role of governments in environmental management is difficult but inescapable. Sometimes, the state tries to manage the resources it owns, and does so badly. Often, however, governments act in an even more harmful way. They actually subsidise the exploitation and consumption of natural resources. A whole range of policies, from farm-price support to protection for coal-mining, do environmental damage and (often) make no economic sense. Scrapping them offers a two-fold bonus: a cleaner environment and a more efficient economy.

Comprehension questions:

Multiple-choice:
Which statement describes the content of the text?

- Economy sometimes mismanages resources with provisions that are meant to support the environment.
- Governments sometimes mismanage resources with provisions that are meant to support the economy.
- Governments sometimes mismanage resources with provisions that are meant to support the environment.
- I'm not sure.

Outcome:
Discarding certain financial supports would be beneficial for economy and ?
(If you don't know the answer, type '?'.)

Summary:
Please give a summary of the text.

A.3 Source code stimuli

A.3.1 Novices

A.3.1.1 L1_SC1

Instructions

Multiple-choice:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be given a MULTIPLE-CHOICE question about the code.

Outcome:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked about the OUTPUT of the code.

Summary:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked to give a SUMMARY of the code.

Stimulus text

```
public class PrinterClass {
    public static void main ( String [ ] args ) {
        System.out.print ( "answer=" ) ;
        System.out.println ( 40 + 2 ) ;
    }
}
```

Comprehension questions:

Multiple-choice:
What is the output?

- answer=42
- answer=
  42
- answer
  =
  42
- I'm not sure.

Outcome:
What is the program's output?
(If you don't know the answer, type '?'.)

Summary:
Please give a summary of the program.

A.3.1.2 L1_SC2

Instructions

Multiple-choice:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be given a MULTIPLE-CHOICE question about the code.

Outcome:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked about the OUTPUT of the code.

Summary:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked to give a SUMMARY of the code.

Stimulus text

```
cake prices are 1.0 and 2.0
for each item
    if cake price is even
        print "even"
    else
        print "uneven"
```

Comprehension questions:

Multiple-choice:
What is the output?

- even
- uneven
- even, uneven
- uneven, even
- I'm not sure.

Outcome:
What is the program's output?
(If you don't know the answer, type '?'.)

Summary:
Please give a summary of the program.

A.3.1.3 L1_SC3

Instructions

Multiple-choice:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be given a MULTIPLE-CHOICE question about the code.

Outcome:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked about the OUTPUT of the code.

Summary:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked to give a SUMMARY of the code.

Stimulus text

```
n = 3
while n > 1
    print n
    n = n - 1
```

Comprehension questions:

Multiple-choice:
What is the output?

- 3
- 3, 2
- 3, 2, 1
- I'm not sure.

Outcome:
What is the program's output?
(If you don't know the answer, type '?'.)

Summary:
Please give a summary of the program.

A.3.1.4 L3_SC1

Instructions

Multiple-choice:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be given a MULTIPLE-CHOICE question about the code.

Outcome:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked about the RETURN-VALUE of 'rect2.area ( )' after the program was executed.

Summary:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked to give a SUMMARY of the code.

Stimulus text

```
public class Rectangle {
    private int x1 , y1 , x2 , y2 ;

    public Rectangle ( int x1 , int y1 , int x2 , int y2 ) {
        this.x1 = x1 ;
        this.y1 = y1 ;
        this.x2 = x2 ;
        this.y2 = y2 ;
    }

    public int width ( ) { return this.x2 - this.x1 ; }

    public int height ( ) { return this.y2 - this.y1 ; }

    public double area ( ) { return this.width ( ) * this.height ( ) ; }

    public static void main ( String [ ] args ) {
        Rectangle rect1 = new Rectangle ( 0 , 0 , 10 , 10 ) ;
        System.out.println ( rect1.area ( ) ) ;
        Rectangle rect2 = new Rectangle ( 5 , 5 , 10 , 10 ) ;
        System.out.println ( rect2.area ( ) ) ;
    }
}
```

Comprehension questions:

Multiple-choice:
The program

- computes the area of rectangles by multiplying their width (x1-x2) and height (y1-y2).
- computes the area of rectangles by multiplying their width (x2-x1) and height (y2-y1).
- computes the area of rectangles by multiplying their width (x1-y1) and height (x2-y2).
- I'm not sure.

Outcome:
What is the return-value of 'rect2.area ( )'?
(If you don't know the answer, type '?'.)

Summary:
Please give a summary of the program.

A.3.1.5 L5_SC3

Instructions

Multiple-choice:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be given a MULTIPLE-CHOICE question about the code.

Outcome:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked about the VALUE of 'currentSpeed' after the program was executed.

Summary:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked to give a SUMMARY of the code.

Stimulus text

```
public class Vehicle {
   String producer , type ;
   int topSpeed , currentSpeed ;

   public Vehicle ( String p , String t , int tp ) {
      this.producer = p ;
      this.type = t ;
      this.topSpeed = tp ;
      this.currentSpeed = 0 ;
   }

   public int accelerate ( int kmh ) {
      if ( ( this.currentSpeed + kmh ) > this.topSpeed ) {
         this.currentSpeed = this.topSpeed ;
      } else {
         this.currentSpeed = this.currentSpeed + kmh ;
      }
      return this.currentSpeed ;
   }

   public static void main ( String [ ] args ) {
      Vehicle v = new Vehicle ( "Audi", "A6", 200 ) ;
      v.accelerate ( 10 ) ;
   }
}
```

Comprehension questions:

Multiple-choice:
The program

- ... defines a vehicle by a producer, that has a type and can reduce its speed.
- ... defines a vehicle by a producer, that has a type and can accelerate its speed.
- ... defines a vehicle by a producer, that has a type and can accelerate and reduce the speed.
- I'm not sure.

Outcome:
What value has 'currentSpeed' after the program was executed?
(If you don't know the answer, type '?'.)

Summary:
Please give a summary of the program.

A.3.2 Experts

A.3.2.1 SC1

Instructions

Multiple-choice:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be given a MULTIPLE-CHOICE question about the code.

Outcome:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked about the RETURN-VALUE of 'rect2.area ( )' after the program was executed.

Summary:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked to give a SUMMARY of the code.

Stimulus text

```
public class Rectangle {
   private int x1 , y1 , x2 , y2 ;

   public Rectangle ( int x1 , int y1 , int x2 , int y2 ) {
      this.x1 = x1 ;
      this.y1 = y1 ;
      this.x2 = x2 ;
      this.y2 = y2 ;
   }

   public int width ( ) { return this.x2 - this.x1 ; }

   public int height ( ) { return this.y2 - this.y1 ; }

   public double area ( ) { return this.width ( ) * this.height ( ) ; }

   public static void main ( String [ ] args ) {
      Rectangle rect1 = new Rectangle ( 0 , 0 , 10 , 10 ) ;
      System.out.println ( rect1.area ( ) ) ;
      Rectangle rect2 = new Rectangle ( 5 , 5 , 10 , 10 ) ;
      System.out.println ( rect2.area ( ) ) ;
   }
}
```

Comprehension questions:

Multiple-choice:
The program

- computes the area of rectangles by multiplying their width (x1-x2) and height (y1-y2).
- computes the area of rectangles by multiplying their width (x2-x1) and height (y2-y1).
- computes the area of rectangles by multiplying their width (x1-y1) and height (x2-y2).
- I'm not sure.

Outcome:
What is the return-value of 'rect2.area ( )'?
(If you don't know the answer, type '?'.)

Summary:
Please give a summary of the program. Tell shortly:

- What is the algorithmic idea?
- How is this idea implemented?

A.3.2.2 SC2

Instructions

Multiple-choice:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be given a MULTIPLE-CHOICE question about the code.

Outcome:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked about the VALUES OF THE VARIABLE 'xy_common' after the program was executed. You don't need to memorize values.

Summary:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked to give a SUMMARY of the code.

Stimulus text

```
import java.util.ArrayList ;
import java.util.List ;

public class Quantities {

   public static List < Double > common ( double [ ] list1 , double [ ] list2 ) {
      List < Double > winners = new ArrayList < Double > ( ) ;
      for ( int i = 0 ; i < list1.length ; i ++ ) {
         for ( int j = 0 ; j < list2.length ; j ++ ) {
            if ( list1 [ i ] == list2 [ j ] ) {
               winners.add ( list1 [ i ] ) ;
               break ;
            }
         }
      }
      return winners ;
   }

   public static void main ( String [ ] args ) {
      double [ ] x = { 2.3 , 8.7 , 9.1 , - 5.6 } ;
      double [ ] y = { - 3 , 0 , 8.7 , 9.1 } ;
      List < Double > xy_common = common ( x , y ) ;
      System.out.print ( xy_common ) ;
   }
}
```

Comprehension questions:

Multiple-choice:
Which statement describes the content of the source code? The program

- ... finds the elements, that are only present in one of the input arrays.
- ... finds the elements, that are in either one of the two input arrays.
- ... finds the elements, that the two input arrays share.
- I'm not sure.

Outcome:
What does 'xy_common' look like after the program was executed with
x = { 2.3 , 8.7 , 9.1 , - 5.6 } and y = { - 3 , 0 , 8.7 , 9.1 }?
(If you don't know the answer, type '?'.)

Summary:
Please give a summary of the program. Tell shortly:

- What is the algorithmic idea?
- How is this idea implemented?

A.3.2.3 SC3

Instructions

Multiple-choice:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be given a MULTIPLE-CHOICE question about the code.

Outcome:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked about the VALUE of 'currentSpeed' after the program was executed.

Summary:
Please read and comprehend the following source code. When you are done, press the left mouse button. Then you will be asked to give a SUMMARY of the code.

Stimulus text

```
public class Vehicle {
    String producer , type ;
    int topSpeed , currentSpeed ;

    public Vehicle ( String p , String t , int tp ) {
      this.producer = p ;
      this.type = t ;
      this.topSpeed = tp ;
      this.currentSpeed = 0 ;
    }

    public int accelerate ( int kmh ) {
      if ( ( this.currentSpeed + kmh ) > this.topSpeed ) {
        this.currentSpeed = this.topSpeed ;
      } else {
        this.currentSpeed = this.currentSpeed + kmh ;
      }
      return this.currentSpeed ;
    }

    public static void main ( String [ ] args ) {
      Vehicle v = new Vehicle ( "Audi", "A6", 200 ) ;
      v.accelerate ( 10 ) ;
    }
}
```

Comprehension questions:

Multiple-choice:
The program

- ... defines a vehicle by a producer, that has a type and can reduce its speed.
- ... defines a vehicle by a producer, that has a type and can accelerate its speed.
- ... defines a vehicle by a producer, that has a type and can accelerate and reduce the speed.
- I'm not sure.

Outcome:
What value has ‘currentSpeed’ after the program was executed?
(If you don’t know the answer, type ‘?’.)

Summary:
Please give a summary of the program. Tell shortly:

- What is the algorithmic idea?
- How is this idea implemented?

A.4 Expert interviews

I - Interviewer
P - Participant

AE22
I: Did you have a certain approach to read and understand these source codes?
P: Ah, there is no methodology that I always follow. I just went, go by what I feel I need, uhm, to grasp more and maybe, maybe that, that there is a clear starting point, I'll start from there, for example eyeing the code and then going to main and starting from there.
I: You said going from main and starting from there. What means 'from there'?
P: Well, for example, if I see a clear path from main, where the program starts from the main and then there is some code call made from there, then I go and see what these calls are about. But, if it's that simple as it was with the rectangle stuff, then it's just that, but if it's more complicated then I have to jump all around the code, because not everything has been saved to my mind, so to speak, at the first try.
I: You had different tasks, like to find a value or write a summary. Did you change your approach somehow according to those tasks?
P: Nnn, not approach, but I would have remembered, I would, should have tried to remember different things, if I had remembered what type of task it was, which I didn't. I just went and tried to comprehend the code and look at important parts.
I: So you didn't really take into account what question will be asked afterwards?
P: Not that much.
I: Ok, and as you said earlier, it's finding the main and going from there. Do you think this approach would be suitable for a novice programmer or somebody who learns programming?
P: I don't think that can be generalized as an answer, yes or no, I mean. I think, if you're, if you have like no idea of what you're doing, then you really can't begin to model a mental model of the program by just eyeing at it. And, you have to be asked a Direct question and the only way to solve that question is to start from somewhere and end at somewhere, so, it, starting from a certain starting point might be a good, or the only way. Maybe. Yeah.
I: Ok.
P: So, I contradicted myself in the beginning. But yeah, that might be.

BE18
I: Du hast Dir eben Quelltext angeguckt und verstanden.
P: Ja.
I: Hast Du dabei irgendeine bestimmte Strategie angewendet?
P: Ähm ... bestimmte Strategie ...
I: Ein bestimmtes Vorgehen ...
P: Also ich hab versucht, quasi, den Fluss des Programms nachzuvollziehen. Erstens mal unten die main-Methode als Einstiegspunkt, und dann hab ich versucht erst mal so grob den Fluss nachzuvollziehen und dann hab ich versucht, die Details zu verstehen, wie jetzt Werte sich verändern.
I: Okay und war das erfolgreich?
P: Ähm ... Ja meist, meistens schon, denk ich, doch.
I: Und könntest Du Dir vorstellen, dass dieses Vorgehen auch für einen Programmieranfänger geeignet wäre?
P: Mh, ja, ich denk schon, also ich weiß nicht genau. Ja doch. Also, ich kenn' es zumindest aus dem Studium auch so, dass man die ganzen, die einzelnen Schritte oder mehrere Schritte aufschreibt, welche, welchen Wert bekommt, bekommen die Variablen, was für einen Namen die haben und ich kann mir schon vorstellen, dass das so für Anfänger gut ist, es so zu machen.
I: Und Du hattest ja verschiedene Aufgabentypen. Mal solltest Du eine Zusammenfassung von dem Programm schreiben oder teilweise auch einfach nur einen Wert angeben. Hatte das irgendwie einen Einfluss darauf gehabt, wie Du vorgegangen bist?
P: Ähm, ja. Also wenn nach einem Wert gefragt wurde, ähm, hab ich's glaub ich eher so ein bisschen umgedreht gemacht, indem ich zuerst zur Ausgabe geguckt hab und dann eher rückwärts geguckt hab, wie

der da rechnet. Also zumindest bei den Aufgaben hab ich das glaub ich so gemacht. Bei den Aufgaben wo ich das Programm verstehen sollte, hab ich's eher so gemacht, dass ich halt erst den Programmfluss versucht hab zu verstehen und dann die Details.

BE26
I: Du hast Dir jetzt gerade eine ganze Reihe Quelltexte angeguckt ...
P: Hm.
I: Hattest Du bestimmte Strategien beim Verstehen?
P: Ähm, na erst mal Überblick verschaffen ...
I: Okay.
P: ... was wirklich aufgerufen wird und in welcher Reihenfolge aufgerufen wird. Also, ähm, erst die, meinetwegen, die public static void main hier und dann gucken was, ähm, die tun.
I: Okay, kannst Du Dir vorstellen, dass diese Strategie auch für einen Anfänger geeignet wäre?
P: Weiß ich nicht. Ähm, weiß nicht. Nee, weiß ich nicht. Kann ich nicht sagen.
I: Okay und Du hattest ja verschiedene Aufgabentypen, also mal Multiple-Choice, mal solltest Du eine Zusammenfassung geben ...
P: Mhm.
I: ... hat das irgendwie einen Einfluss gehabt, wie Du durch den Quelltext gegangen bist?
P: Nein.
I: Nee?
P: Nein.
I: Okay.
P: Außer halt wenn vorher angesagt wurde, achte auf den Wert von XY, dann hat man sich da drauf konzentriert.

BE29
I: Also Du hast gerade angefangen, dass Du bei den ersten Quelltexten was anderes gemacht hast von der Strategie her.
P: Genau, an den ersten Quelltexten habe ich erst versucht, so irgendwie, äh, mir die ganze Klasse anzugucken, zumindest also den kompletten Quelltext und ähm mir im Groben erst mal anzugucken, wie ist es aufgebaut, ähm, hab dann irgendwann festgestellt, dass es mir nicht weiterhilft ...
I: Okay.
P: ... sondern dann bin ich doch nach unten zur main-Methode, hab geguckt, "Okay, die ruft die Klasse auf, guckst Du Dir mal Klasse an, ah ok, die rufende Klasse wird ja auch noch benutzt" und dann hatte es da besser funktioniert. Also bei der ersten Aufgabe hatte ich es wirklich so versucht und dann ging auch und deswegen dachte ich, und die zweite war glaube ich auch die rekursive, das war die mit der Sortierung, das weiß ich noch ...
I: Mhm.
P: ... ähm, da bin ich dann gleich in die main-Methode, weil ich dachte "Oh Gott, nicht dass Du da wieder so lange davor sitzt, guck mal lieber, dass Du Dich Schritt für Schritt da durchhangelst".
I: Würdest Du vermuten, dass Du das normalerweise auch machst, also in deiner täglichen Arbeit auch erst so nach einer main suchen, oder gehst Du da ...
P: Na 'ne main gibt's dort nicht, die haben wir jetzt so nicht ...
I: Okay.
P: ... nee, da gucke ich mir eigentlich immer die Modelle an, also so viel Algorithmik betreiben wir auch halt nicht, also da geht's eher doch um die saubere Modellierungen und so.

CO20
I: Okay, Du hast eben mehrere Quelltexte gelesen und verstanden. Hattest Du irgendein bestimmtes Vorgehen nach dem Du jedes Mal vorgegangen bist? Also das Du jedes Mal hattest?
P: Also zuerst die main-Methode, dann den Constructor meistens, um die Übergabe der Eingabeparameter anzusehen und dann die Methodenaufrufe.
I: Okay, und Du hattest ja auch verschiedene Aufgabentypen, mal solltest Du nur einen Wert ausgeben, mal solltest Du eine algorithmische Idee finden. Hat es einen Unterschied gemacht, wie Du vorgegangen bist?

P: Nein, ich habe meistens immer auf die main-Methode geguckt und danach eben nach Funktionsaufruf, bin ich den Funktionen gefolgt.
I: Okay, und kannst Du Dir vorstellen, dass das für einen Programmieranfänger auch geeignet ist, dieses Vorgehen?
P: Der Vorteil ist, dass man überflüssige Informationen rausfiltert, schnell. Finde ich.
I: Also ja oder nein?
P: Ja.

HI27
I: Du hast ja grade mehrere Quelltexte gelesen und verstanden. Hattest Du dabei ein bestimmtes Vorgehen?
P: Das ist doof, weil ich schon mal gesehen habe, wie andere Leute Quelltexte lesen und mir die entsprechenden Daten angeguckt habe, deswegen bin ich da ein bisschen subjektiv, aber ich habe schon versucht immer zuerst die Hauptfunktion also main-Funktion zu finden und zu gucken, welche Funktionen es gibt und wie die Hauptstruktur ist. Wann welche Funktion von welcher anderen Funktion aufgerufen wird, was die Eingabe- und Ausgabeparameter der Hauptfunktion sind und dann, was die aufgerufenen Unterfunktionen so machen. Also einen top-down Ansatz.
I: Okay. Und Du hattest ja verschiedene Aufgabentypen. Mal solltest Du eine Zusammenfassung schreiben, mal nur einen Wert angeben. Hat das irgendeinen Einfluss drauf gehabt, hast Du dein Vorgehen angepasst?
P: Nein, die waren eigentlich sehr ähnlich, weil das war ja immer. Oder ich hab mich schon, also wenn zum Beispiel die Aufgabe war, erinnere Dich an den Wert einer Variablen, dann habe ich eigentlich zuerst geguckt, wo diese Variable auftaucht und was in diese Variable reingeschrieben wird und dann habe ich geguckt wo ist da die Funktion, die das macht, was in die Variable reingeschrieben wird und so was. Und ich habe zuerst, ich hab schon gesucht, zielgerichtet nach dem geguckt, was am Ende rausgegeben wird oder was gefragt wird.
I: Okay, kannst Du Dich daran erinnern, als Du die Multiple-Choice Questions hattest. Bist Du da irgendwie anders vorgegangen als jetzt?
P: Ich glaube nicht. Nein.
I: Okay. Und könntest Du Dir vorstellen, dass das Vorgehen das Du am Anfang meintest, erst mal die main suchen und von da aus weiter gehen, auch für einen Anfänger, also einen Programmieranfänger, geeignet wäre?
P: Ja.
I: Weil?
P: Weil man sich so am besten einen Überblick über das Gesamtsystem verschaffen kann und sich mit den Details erst später beschäftigen kann. Und dann vielleicht auch unnötige Sachen, also sich nicht von Anfang an auf unnötige oder unwichtige Sachen fokussiert.

IE30
I: Okay, Du solltest ja jetzt eben Quelltexte lesen und verstehen.
P: Mhm.
I: Hattest Du dabei irgendeine bestimmte Strategie, die Du verfolgst hast?
P: Also ich versuche in der Regel so den, den, den, Ablauf nachzuverfolgen. Also ich meine, ich fang, ich weiß, dass ich selber, in der Regel erst mal anfange oben, na, so erst mal gucken, weil erst mal ist das Text und dann sag ich "Okay, aha, okay, Name, Klasse, usw." und dann guck ich aber in der Regel eher bei der main-Methode nach, "Was macht denn die eigentlich?". Und arbeite mich dann halt von da vor, also das ist so strukturell eher so. Mal gucken, also was macht die, was ruft die auf, was ruft dann das auf und so weiter und sofort und versuche da anhand dessen den, den Ablauf nachzuverfolgen.
I: Alles klar. Und könntest Du Dir vorstellen, dass dieses Vorgehen auch für einen Anfänger geeignet wäre?
P: Ähm, also ich meine, ich denke, es ist generell schon 'ne gute Idee das zu machen, also es ist halt so, herauszufinden, wo der Einstiegspunkt in das Ganze ist und dann was zu machen, aber dazu muss man natürlich, also wenn man als Anfänger weiß wo das ist, könnte ich mir schon vorstellen, dass das was bringt. Wenn man den Text einfach erst mal so von oben nach unten durchliest, das bringt in der Regel nicht so viel, weil es ja nicht den logischen Ablauf widerspiegelt.
I: Okay.

P: Also meine Meinung dazu.
I: Und dann hattest Du ja verschiedene Aufgabentypen. Mal solltest Du eine Zusammenfassung schreiben oder ...
P: Ja.
I: ... mal auch einen Wert ausgeben.
P: Mhm.
I: Hat es irgendwie einen Unterschied für Dich gemacht, wie Du vorgegangen bist?
P: Ähm, nee, also der Grobablauf ist dasselbe, man muss halt rausfinden was macht das Programm eigentlich.
I: Mhm.
P: Aber, ähm, also zum Beispiel bei dem Wert war ich mir am Schluss gar nicht mehr so sicher, "Moment mal, war das jetzt größer oder kleiner?". Also das wär so'n Ding wo ich dann am Schluss nochmal hätte nach, eigentlich nochmal zurück hätte gehen müssen und nochmal nachgucken müssen.
I: Okay.
P: Also ich hatte gesehen, es ist sortiert, aber ob es aufsteigend oder absteigend ist, das hätte ich jetzt am Schluss gar nicht mehr so genau gewusst.
I: Okay.
P: Also weil es ja auch, ja, so'n, so'n, so'ne relativ kompakte Sache ist. Also was man in der Regel, ähm. Also jetzt würde ich sagen, ist eigentlich das normale Vorgehen, man schreibt 'nen Unit-Test der sagt: "So, ich will wissen, hinterher soll es so aussehen" und dann gucken wir ob der Code das macht. Und wenn nicht, guckt man nochmal genauer hin warum er es nicht macht und dann dreht man vielleicht das Kleiner um oder setzt ein Größer oder umgekehrt, je nachdem was man braucht. Aber ja. Also das ist noch was anderes als wenn's darum geht, "Wie ist allgemein der Algorithmus?". Wenn man da sieht "Ah ja, okay, hier haben wir Rekursion, hier haben wir 'ne Schleife, hier machen wir dieses, hier machen wir jenes". Es ist schon noch irgendwie ein kleiner Unterschied, aber nicht zum allgemeinen Ablauf, sondern im Detail.

KK24
I: Did you have a certain approach to read and understand these little programs?
P: Not the particular approach, but I first follow the main-method and what it's trying to do. And then I looked in the method that it calls and after that I did. What about it? I, I really don't understand.
I: Ok, well, you said you're looking for the main?
P: Yes, main-method.
I: And then, how did you proceed from there?
P: There you can find the actual method that it's calling.
I: Ah, okay.
P: And then I followed the method and then the output.
I: You had different types of questions, like what value is that or please write a summary. Did you change your approach according to the task?
P: No, not really. I followed the same approach.
I: Ok, and would you think that this approach is also okay for a novice?
P: For novice?
I: For someone who just learns programming.
P: Might be. I don't know, because everyone has their own approach, how they follow that. I don't really know.
I: So, you think every person has their own approach?
P: Yes.
I: There is not the general one that you could apply to everyone?
P: I don't think there is a general approach that can be followed by everyone.

LK23
I: You just read and understood sample source codes. Did you have some kind of approach to go through those?
P: Yes. I always would go, I always will quickly first go through the classes, over the functions and then I would immediately go to the main and to see actually what are the inputs. Actually mostly to understand what are the arrays are for and stuff. So, and then when I've seen what is in main required,

then I come back to the functions and or classes and then see what they are for, and then, I mean, try to think.
I: Ok. And you had different tasks, like find the value or write a summary about the algorithm. Did that change your approach?
P: Maybe just that, when it is value, so I would be more careful to see and calculate in mind. What the output should be and when it's summary, I don't really care, cause summary that didn't really say, I mean doesn't require to say how much, the results should be, I mean, output should be.
I: And do you think that approach would also be ok for a novice or someone who learns programming?
P: I, yeah, I think actually it's a kind of the best in a way, but I think that maybe somebody new, he will spend more time first to go through functions and then only after some time to come to main. But actually, I believe this is not. Actually I really think that the coming from main is more important and makes more sense, because the main give you real data so you actually knows what arrays supposed to be, physically. Then it's easier to read code. Cause you are more expecting, how you, what you are doing with that array or whatever, or that e.g. indexes.

MR05
I: Du hast ja gerade mehrere Quelltexte gelesen und verstanden. Hast Du dabei irgendeine bestimmte Vorgehensweise gehabt, irgendeine Strategie, nach der Du vorgegangen bist?
P: Ähm,von meinem Gefühl her, ich starte beim Klassennamen ...
I: Okay.
P: ... die, wenn sie sprechend benannt ist, schon mal einen ersten Hinweis gibt, was mich erwartet.
I: Okay.
P: Ähm, der erste Ansprungspunkt wäre hier immer die immer die main-Methode ...
I: Mhm.
P: ... um zu schauen was würde überhaupt passieren um von da aus dann sukzessive in die Methoden reinschauen.
I: Okay. Ähm, denkst Du, dass das gleiche Vorgehen auch für einen Anfänger geeignet wäre?
P: Ja.
I: Ja? Kann man so abarbeiten?
P: Ich denke schon. Das ist so ein bisschen der Flussreihenfolge folgen.
I: Mhm. Fluss, meinst Du jetzt Kontrollfluss oder Datenfluss?
P: Ähm, ja so vom Kontrollfluss her, vom Status der main-Method her, wie sind die ersten angesprochenen Methoden zu verstehen, was dort passiert und wenn es dort wieder Unterbrechungen gibt, vielleicht die zwei Möglichkeiten, entweder ich versuche erst mal eins zu verstehen oder ich gehe gleich tiefer in die anderen rein.
I: Okay und ähm Du hattest ja verschiedene Aufgabenstellungen. Mal solltest Du nur die Ausgabe finden bzw. solltest die algorithmische Idee vom Ganzen beschreiben, hatte das irgendeinen Einfluss darauf gehabt wie Du vorgegangen bist?
P: Ähm, ich denk mal, wenn ich 'ne Zusammenfassung geben sollte, hab ich mir mehr Mühe gegeben, die einzelnen Zwischenschritte zu merken ...
I: Ah, okay.
P: ... währenddessen das beim Ergebnis, das jetzt nicht ganz so wichtig war. Wenn die Frage kommt, wie war das implementiert, sind vielleicht doch ein paar detailliertere Informationen wichtig gewesen.

PA24
I: Did you have some kind of approach how to read and understand the source codes?
P: Yeah, I usually start from the main program and then see what that does. Like see what gets put in to what and then after that I follow the path of the, like executing the program in my head, kind of. Seeing what you actually need to read and then lazily only reading those parts. Of course in these you will always read everything, but in most normal cases, you wouldn't need to read all the classes to understand the program, you just need the ones that are actually executed so. And I have troubles with recursion, because it calls itself and I'm really bad at guessing the, like, ending parameters and at last one I was really confused, because it just seem, it just copies the values inside the one array and does nothing sensible, but that might have been just a misunderstanding on my part. But yeah.
I: Ok, but you had different kinds of tasks, sometimes you just had to say the value and sometimes you were asked to write a summary. Did that change your approach?

P: Uhm, not really, unless the code was really self-explanatory and then I could, I didn't have to even go so far that I would need to analyze the program. For example the rectangle one, I just like saw, ok, this will describe a rectangle and than I just scanned it for traps like, so that there is nothing wrong hidden in the code, so it's clean, and then I just came to the conclusion that, ok this is probably just a rectangle. Nothing more than that. And then I, uh, yeah, I checked the values that were put in the main, so if there was anything, uh, that would be related to the values, so I would know the values of the rectangles. Like, just in case. So, just checking the main for how the program works and then doing the summary. Because the, the class itself, kind of already looked that it describes a rectangle. Uhm, it was like self-explanatory.
I: Ok. You said first that your approach is to first look at the main and then follow the execution order. Do you think this approach would also be suitable for a novice, like someone who just learns programming?
P: Yes. I think. It might be like time-consuming to get like the certain mentality, like what to memorize and what not to memorize when doing this. But I think, in my case I really like procedural code, because it's how the computer really runs the code, even though you try to have higher level like objects and stuff like that, but still the execution in almost everything is kind of procedural so, it, in. I think it's an easy approach for programming to get procedural stuff and then like extend that to the abstract level of like objects and stuff like that. Like, it's still kind of procedural. I think, it's easy for the brain to do stuff in sequences.

SI28
I: You just read and understood some source codes. What was your approach to read and understand those?
P: I guess, I started from the exit point. For example from the main method, then backwards, looking for the, what the, how the program is executed. Yes, something like that.
I: And you had different task types, like find the value or write a summary. Did you change your approach according to that?
P: I don't think so. I just follow mine, my mind. Maybe that's my habit to read some strange source code. I follow my own logic.
I: Ok. And do you think a novice programmer, somebody who just learns programming could also use this approach?
P: Yeah, I think so. It's kind of somehow a fixed matter how to understand a program, Java program or whatever, some.

TU15
I: Okay, Du hast ja grade mehrere Quelltexte gelesen und verstanden. Hattest Du dabei ein bestimmtes Vorgehen?
P: Ja, ich habe zuerst so grob geschaut. Ich habe zu erst grob geschaut, aber dann habe ich immer zuerst diese main-Methode gesucht, wo sie überhaupt ist. Und so ein bisschen geguckt welche Teile vom Source Code, denke ich, hab geguckt, welche Teile vom Source Code überhaupt relevant sind, um einfach zu sagen: okay, die muss ich nicht lesen. Aber bei den Aufgaben waren alle relevant. Aber ich habe mit der main-Methode angefangen und so ein bisschen geguckt, wo die Teile landen und ich habe auch versucht mit dem Beispiel mit zu arbeiten, so ein bisschen im Kopf zu rechnen, statt nur auf das Algorithmus zu gucken.
I: Und es gab ja verschiedene Aufgabentypen. Mal solltest Du einen Wert ausrechnen oder einen Summary schreiben. Hat das einen Unterschied für Dich gemacht? Also hast Du dein Vorgehen angepasst?
P: Beim Ergebnis habe ich dann wirklich nur drauf geschaut, was erreiche ich jetzt. Ich habe nicht wirklich darauf geachtet welche Algorithmen dahinter stecken, sondern einfach so grob erkannt, kenne ich das Pattern oder nicht, aber einfach nur schnell nach dem Ergebnis gesucht. Und bei Summary habe ich einfach nicht erwartet, konnte ich nicht erwarten, welche Art von Frage kommt und da habe ich bei einer sogar geguckt: Okay wie heißen die Variablen jetzt hier, weil vielleicht wird auch gefragt, war das CamelCase. Ich wusste einfach nicht was da in Frage kommen kann. Das hat mich dazu gezwungen gründlicher zu lesen, aber ich weiß nicht, ob ich dann immer. Ich glaube mit dem Beispiel war ich sofort schneller drin, um zu wissen, was passiert.
I: Du meintest ganz am Anfang, Du hast erst nach der main-Methode geguckt und bist dann so ein bisschen im Prinzip die Ausführung lang oder hast das Beispiel ausprobiert. Kannst Du Dir vorstellen,

dass das für einen Anfänger auch geeignet ist, so ein Vorgehen.
P: Kann ich nicht beurteilen. Das ist schwer. Ich kann mir vorstellen, dass es schwer ist sich eine mentale Karte von Source Code zu bauen im Kopf. Und manchmal, je nach Aufgabe, kann das sein, dass viele Teile überflüssig sind und daher ist dann diesen kompletten Source Code lesen eher nicht geeignet, würde ich sagen, aber vielleicht machen es Anfänger. Aber ich habe versucht zu gucken, welche Teile sind überhaupt relevant.
I: Die Überlegung ist eher, sollten wir versuchen den Anfängern die gleiche Strategie beizubringen, die Du als Experte hast. Deswegen frage ich. Ob Du meinst, dass dein Vorgehen vielleicht auch schon geeignet wäre, dass man es einem Anfänger beibringt.
P: Vielleicht. Ich habe es gewiss einfacher für mich, weil ich einfach einem konkreten Beispiel folgen kann und so ein bisschen gucken was passiert und dann erkenne was ist relevant überhaupt zum Lesen. Das ist vor allem bei größeren Source Code Beispielen, kann man einfach nicht alles lesen. Nicht mal anfangen, also lieber nicht.

UL29
I: Du hast ja jetzt Quelltexte gelesen und verstanden. Hast Du dabei eine Strategie angewendet?
P: Ja, pff, in dem Sinne, dass ich mir zuerst nur die main-Methode angeguckt habe und dadurch nicht so von oben nach unten gelesen habe, sondern eher mir erst mal geguckt, was wird als erstes aufgerufen und dann Schritt für Schritt das Programm durchgegangen bin.
I: Okay. Und könntest Du Dir vorstellen, dass diese Strategie auch für einen Anfänger geeignet ist?
P: Ähm, sicherlich, ja, definitiv. Also man muss ja irgendwie nicht gleich das ganze Modell verstehen, und wenn man dort verstanden hat, und dann guckt man was es eigentlich macht ...
I: Mhm.
P: ... sondern man guckt erst mal, "Ah, was macht's denn" und danach kann man sich ja nochmal angucken wie es implementiert wurde und ob das gut oder schlecht ist.
I: Okay. Du hattest ja verschiedene Aufgabentypen, mal solltest Du ne Zusammenfassung schreiben, oder mal solltest Du einen konkreten Wert angeben. Hat es für Dich einen Unterschied gemacht wie Du dann vorgegangen bist?
P: Mhh, nee.
I: Gar nicht?
P: Nö.