



Analysis of Code Reading to gain more Insight in Program Comprehension

Teresa Busjahn, Carsten Schulte
Department of Computer Science
Freie Universität Berlin, Germany
{busjahn,schulte}@inf.fu-berlin.de

Andreas Busjahn
HealthTwist GmbH, Germany
busjahn@healthtwist.de

ABSTRACT

Code reading, although an integral part of program comprehension, is rarely reflected. In this paper, we want to argue for a research approach and direction exploiting the potential that lies in the analysis of reading processes. Based on the vast experience compiled in psychology and some studies in computing, eye tracking and think aloud were elaborated as a viable research instrument for code reading studies. We conducted a feasibility study, designed to examine the actual process of code reading as the sensory starting point of comprehension. Computational and statistical tools were developed to facilitate data capture and analysis for eye tracking experiments. Results do not just provide proof of concept but already emphasize differences between reading natural language text and source code, as well as a distinct attention allocation within different code elements like keywords and operators.

In conclusion we suggest a combination of theory-driven selected stimuli material, a carefully designed procedure of eye tracking, complemented with suitable post-tests on comprehension as well as retrospective think aloud in order to obtain additional information on the linking process between perception and comprehension. As an addition to other research approaches this should most certainly help us to improve our knowledge of comprehension within an educational research framework.

Categories and Subject Descriptors

K3.2 [Computers & Education]: Computer and Information Science Education – *computer science education, information systems education*.

General Terms

Experimentation, Human Factors.

Keywords

CS Ed Research, Educational Research, Code Comprehension, Program Comprehension, Eye Tracking, Code Reading

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Koli Calling '11, November 17--20, 2011, Koli, Finland.
Copyright 2011 ACM 978-1-4503-1052-9/11/11...\$10.00.

1. INTRODUCTION

Program comprehension (PC) is an important element of computer science education, taking up a considerable amount of time and mental resources of programmers. Most of the relevant research in this field starts with components like internal representation of knowledge, or internal inference as a means of debugging. In our empirical study we decided to take one step back and put the focus on the very first phase of PC, i.e. code reading, since it is an essential part of programming and the basis for the components mentioned above. Learning a programming language, development of software and the various tasks required in its maintenance necessitate the reading and understanding of already written code. Furthermore, the design of IDEs and tools to facilitate programming would certainly benefit greatly from a better understanding of the reading process [1].

Previous studies in this context commonly used think aloud and reports of programmers (von Mayrhauser & Vans (1994) [18]). These methods are to some extent useful to analyze PC, but they influence the cognitive processes and lessen the comprehension capacity. Studies that aim at the understanding of natural language text and images often use records of eye movements, following the rationale that they provide a fundamental insight into the cognitive processes. However there are only few projects that studied eye movements during code reading. The results of the few conducted studies with source code (SC) indicate that eye tracking especially in combination with retrospective think aloud (RTA) is a promising approach for the analysis of the cognitive processes during PC.

The paper is organized as follows: An overview of the selected methods is followed by a review of eye tracking research with regard to program comprehension. Subsequently a refined study design is presented that is adequate to explore PC.

2. RESEARCH ON READING AND PROGRAM COMPREHENSION

2.1 Introduction to eye tracking as method

The choice of methods depends on the component of the comprehension process to be analyzed. Reading can be analytically divided into perception and comprehension. Eye tracking is a means to observe the process of perception in order to obtain insights in the process of comprehension. Measurements like eye tracking provide very objective data by recording the actual visual behavior. Experiments with different techniques obtained comparable results, thus corroborating the quality of eye tracking [8].

For example, Bednarik and Tukiainen [3, 5] studied if less complex instruments like the restricted focus viewer (RFV) provide comparable depth of information. An RFV is a visual attention tracking system that only allows a small region to be seen in focus and blurs the rest of the display. The subject moves a mouse over the region of current interest. In order to survey the effect of using an RFV to trace visual attention during Java program debugging, the RFV was tested on novices and experts. The mouse movements during blurring were recorded, together with the eye movements, so attention switching and fixation durations could be analyzed. The restricted view seems to influence the cognitive behavior. Attention switching and the mean fixation duration decreased in both groups, this effect is more substantial for experts. Furthermore, the eye tracking device registered more switches between code, visualization and output. Nevertheless the blurring by the RFV caused no significant difference in debugging performance or the distribution of time spent on the three different areas. These study results indicate that information about visual attention allocation gathered by the RFV may be seriously biased by interference of measurement and object of investigation. The eye tracking data gained during the study proved to be more objective, supporting the suitability of eye tracking for recording visual attention on an unobstructed display.

A method frequently used in comprehension studies is concurrent think aloud (CTA). This procedure reveals internal comprehension processes, but it is hard to accomplish for the subjects and influences their behavior, as it imposes additional cognitive load: Since the test person cannot fully concentrate on the task, the mental capacity is reduced. Another problem of methods like CTA and self-reports is that different subjects may give varying descriptions of the same processes, thus they are very subjective. An improved variation of think aloud is the combination of eye tracking with RTA. The test subject completes the task without disturbance while the eye movements are recorded. Afterwards the recording is shown to the subject, asking for comments. Thereby the respective specific advantages of eye tracking and CTA can be combined. Questions like 'What was looked at, but not really seen' can be answered this way. The RTA data often includes more information about cognitive operations, while subjects during CTA mainly give manipulative statements like 'I clicked ...'. Comments during RTA are usually longer and more frequent, and have a higher quality, probably due to the absence of the double load of task solving and verbalizing the thoughts ([14]).

Important data obtained during eye tracking are the number, duration, and the sequence of so called fixations. While looking at pictures or text, the eye stays on one point for a fraction of a second and then moves quickly to the next position. These holding points between eye movements are called fixations, during reading they usually last between 200 and 400 ms. The specific movements that bring the eye on a certain spot are called saccades, they take 20 to 120 ms depending on the distance to cover. There is no intake of visual information during the saccades. Backwards movements in the text are called regressions. 10 to 15 % of the fixations are regressive. Good readers are characterized by few regressions and short fixations.

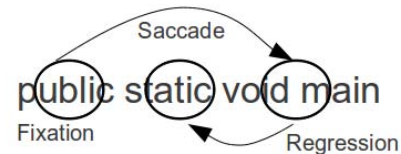


Figure 1 Eye tracking measurements

In the center of the fixation, the area of highest visual acuity, three to four characters are processed, left and right to that region 3 to 15, depending on reading direction. The processing gets more superficial with distance to the fixation. These parameters depend on several factors, like text difficulty and formatting.

Present research results indicate relations between eye movements and cognitive comprehension processes. However these are complex and there is no easy matching. Fixation duration is usually associated with cognitive effort, so difficult texts induce longer fixations, short saccades and frequent regressions. The frequency of a word influences how long it is fixated, but fixation duration should not be confused with the amount of time needed for processing [6–9, 12, 20, 26].

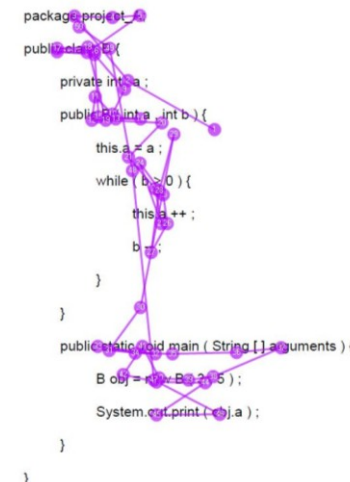


Figure 2 Sample visualization of eye tracking data (by Tobii Studio)

Unlike e.g. the ear, the eye is an active receptor organ; it has to be directed at a stimulus. Measuring direction of eye movements is therefore a proxy indicator for attention.

The eye provides a plethora of information about perception and in addition is easy to access. Multi-channel recording of several parameters is possible, exploiting this rich source of information. Today's recording devices have a high accuracy and can be used in true-to-life situations [6]. There are several measurement principles, as e.g. exploiting the corneal reflex. The relation of the reflection of a light source on the cornea and the pupil are used to determine the position of the eye. This requires a calibration process before the actual recording [6, 7, 12].

Eye tracking is "high tech", but has advantages over methods like CTA and observations: „these techniques are much easier to implement than a study of eye movements, the data obtained from them may be viewed as providing interesting suggestions about what is going on in normal silent reading. However, one can never be sure; if you want to know what happens in normal

silent reading, the best thing to study is normal silent reading” [20], p.185. The same applies to PC, if you are interested in normal undisturbed code reading, you have to study exactly that.

2.2 Previous work on Program Comprehension with eye tracking

There is a substantial body of work regarding code reading, tracing, and writing [16, 17, 25] as well as comprehension from an educational perspective [15, 18]. While a full review of that work is not in the scope of this article, we briefly discuss the research perspective of studies using eye tracking [1, 2, 4, 10, 11, 13, 23, 24, 27].

One of the earliest studies was Crosby & Stelovsky, 1990 [10]. The study sought to analyze differences between reading non-procedural prose and program text and the influence of programming experience on reading. Materials used in the study were a Pascal program with an implementation of binary search, including inline comments and additional visualizations of the program execution. Ten novices and nine experts were studied. Crosby and Stelovsky’s results show a difference between the text types: reading source code caused a higher number of fixations and more regressions. However, that comparison was done using results of a different study using natural language text, there was no examination of subjects’ reading skills.

Code elements were divided into categories with diverging complexity: comments, comparisons, complex statements, simple assignments and keywords. Both, novices and experts were fewest interested in keywords and simple assignments, while comments got the most attention. Novices looked more at comments and comparisons, experts at complex statements. The authors concluded that comments and complex statements contain most information about the algorithm, whereas keywords are of low semantic value. This interpretation is somewhat problematic, as number and length of fixations have not been adjusted to the length of the words or number of elements fixated. Since comments hold the most characters, it is not surprising that they need longest to be read.

Another study result refers to reading patterns. Subjects tended to read from left to right and from top to bottom, but differed in the number of passes. The patterns range from a single scan, with concentration to certain areas to periodic scans with focus to different regions. Additionally, there was a comparative strategy, where the comparison of areas superimposed reading in natural order. It was not possible to detect reading patterns that distinguished novices from experts. The two most similar patterns even were from an expert and from a novice.

Overall the study revealed several interesting insights, but interpretation regarding differences between natural text and programming text and the influence of programming experience are difficult. The combination of source code, natural language text comments and visualizations aims rather at the integration of visual information from different sources than at code reading. As data on natural text was obtained outside the actual study, no real comparison is possible between the reading of source code and prose. In general the study was conducted somewhat unsystematic, e.g. participants had different prior knowledge of the used algorithm (novices were taught exactly that implementation of binary search two month before the experiment).

In context of studies on beacons, the data of the 1990 report were reevaluated [11]. The mapping of fixations to categories was normalized, by dividing the number of fixations by the

number of words / characters in each category. This form of normalization remains a compromise, since neither characters nor words correspond to a unit that can be perceived with one fixation. In contrast to the former report, comments no longer get the most fixations, both novices and experts mainly concentrate on complex statements instead (Figure 3).

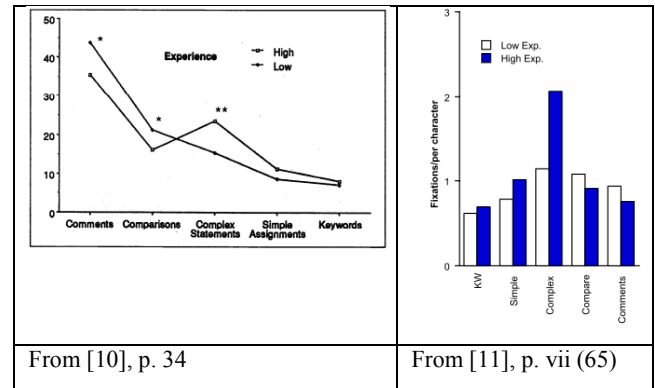


Figure 3 Attention allocation in different element categories, stratified into low and high experience

In a study presented 2006, Aschwanden and Crosby [1] aimed at an evaluation of the concept of beacons in program comprehension. 12 java programs, which implemented six different algorithms either recursively or non-recursively were used as stimulus material. The 15 participants were CS students in their third year.

Prior to the eye tracking part of the study, a pre-test gathered information on experience and interest in programming. A post-test was used to get information on the difficulty of the texts, and comments on the experimental setting. In the post-test subjects were also asked to identify beacons (the most important parts of the source code). But the study results did not show any correlations between the lines nominated most important and those that were looked at most. Questioning subjects about the essential code areas is not a reliably method, since the named lines do not agree with those that received most visual attention. Eye tracking methods are therefore more objective. As preliminary results authors interpreted those parts as beacons that got fixations of at least 1000 msec.

Uwano et al. [23, 24] used eye tracking in the context of software-review approaches. Six C-programs with a logical error served as stimulus in the study with five subjects. The analysis revealed a typical perception strategy, called scan-pattern: Before concentrating on certain parts of the program, the subjects initially read (scanned) the whole text sequentially from left to right, top to bottom. A more thorough scan correlates with quicker identification of the error.

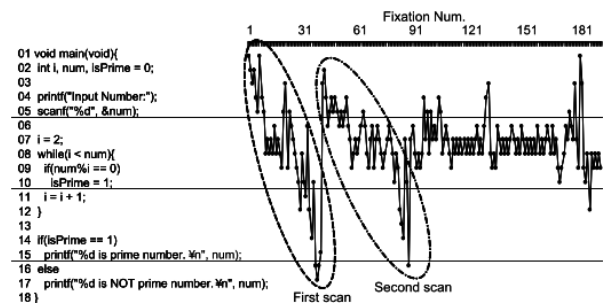


Figure 4 Visualization of scan pattern (from [24], p. 323)

First interview (with source code only)	Second interview (with source code and eye movements)
· I thought something is wrong in the second while loop. · Firstly, I reviewed main function, and then, read the another one. · I simulated the program execution in mind assuming an input value. · I checked the while loop for times.	· I did not care the conditional expression of the loop. · I watched this variable declaration to see the initial value of the variable. · I thought this input process was correct because it is written in a typical way. · I could not understand why this variable is initialized here.

Figure 5 Comments during post-test (from [24], p. 326)

Uwano et al. used an interesting methodological variation: In some of the post-test RTA interviews only the program text was presented, in the others program text together with a visualization of the eye movements of the subject (Figure 5). Interestingly, the playback of eye movements triggered more details and code-specific comments. The authors concluded that eye movements can give some information about the reviewer's cognitive process. A shortcoming of this study is the missing check for the reading skills of the subjects, as some results concerning reading time could be due to different overall reading skills.

Bednarik and Tukiainen [2, 4] used java-texts together with visualizations as material in their eye tracking studies. Both studies had 18 participants. For the analysis of the earlier study, they implement and discuss a fine-grained methodology, in which data were divided into temporal segments. For each of the five segments / phases they counted the number of switches between a) code and visualization, b) code and program output, and c) visualization and output. No differences were found between novices and intermediates concerning switches between code and visualization. One finding that correlates with experience is that the more experience a programmer has, the less likely he/she is to use the visualization. Experienced subjects read the code and used the visualization to validate their hypothesis, while novices first employed the graphical representation to understand the program.

In a similar experiment they studied debugging with source code

that contained non-syntactic errors. Experts again focused first on the code and later on the output, novices preferred the visualization altogether. In later phases of the debugging process a concentration on the output correlated with a higher number of detected errors. Using smaller units of comparison reveals significant differences, while overall few differences between novices and experts could be found. Bednarik and Tukiainen emphasize the need for methodological advances in data analysis.

The approaches of Guéhéneuc 2006 [13], and Yusuf et al. 2007 [27] are complementary to the presented studies, as they focus on visual text, namely UML.

2.3 Conclusion

The presented studies illustrate the usefulness of eye tracking for studying PC. Despite some methodical shortcomings, the designs provide some insights and can be used as starting point for designing a new study.

It is surprising that seemingly none of the above mentioned studies examined their participants reading ability in natural language text. Since those studies were conducted with a small number of subjects (see Table 1), a dyslexic could seriously distort the data. Without knowledge of the participants reading behavior, no real comparisons are possible, e.g. long fixations might be normal for a certain person and therefore not induced by the stimulus material.

Table 1 Comparison of conducted eye tracking studies

Study	Participants	Experience levels	Methods	Stimuli programs				Language	Comparison with NT
				algorithms	number	bugs	LOC		
Crosby & Stelovsky (1990) [10]	19	Novices Experts	Eye tracking	Binary search	1	yes	10	Pascal	yes
Aschwanden & Crosby (2006) [1]	15	Intermediates (recruited from a third year computer science class)	Eye tracking	Sum of nbrs in array / a ^{b/} Factorial computation / Binary search / GCD / Fibonacci-nbrs (each recursive and non-recursive)	12	no	n/a	Java	n/a
Uwano et al. (2006) / Uwano et al. (2007) [23, 24]	5	Experts	Eye tracking RTA	Sum-5 / Accumulate / Average-5 / Average-any / Swap / IsPrime	6	yes	12-23	C	n/a
Bednarik & Tukiainen (2006) [2]	18	Novices Intermediates	Eye tracking	Recursive binary search / Naive string matching	2	no	34-38	Java	n/a
Bednarik & Tukiainen (2008) [4]	18	Novices Experts	Eye tracking	n/a	3	yes	Tens of lines	Java	n/a
Feasibility study	15	Novices Intermediates Experts	Eye tracking RTA	Switch-operation / Palindrome / IsLetter / IsSorted / Average / Sum (iterative) / a ^{b/} GCD / Min/Max of array / Reverse elements in array	10	no	10-21	Java	yes

3. STUDYING CODE READING

In this section we present an example of a feasibility study combining eye tracking and think aloud for a comparison of reading natural text (NT) and program text.

3.1 Study Design

The study uses a combination of eye tracking and retrospective think aloud, although the RTA data have not been analyzed yet.

The study design was developed involving psychologists specialized in reading studies. As programming language we chose Java, because of its wide use and representativeness. For the last ten years it has always been on one of the first three spots of the Tiobe Programming Community Index and more than 50 % of the languages used are object-oriented [22].

Using the previous studies as basis, an initial set of 11 small programs with a varying complexity was developed as stimuli. For every text a multiple-choice-question about the program's function was formulated. The programs cover fundamental concepts, like loops and branches. Since the frequency of a word influences eye movements, the appearance of every Java keyword was counted in the JDK 6. To cover a large spectrum, keywords with different frequencies from rare to common were included. In order to avoid artifacts from a certain programming style, some variations were put on the code. Names of variables and methods are meaningful, but not to descriptive of the program's function, like 'num_array'. Coding style was varied between texts in order to avoid artifacts from repetition and technique.

In a pre-study the initial code examples were tested on five novices and five experts. In a questionnaire their programming ability was assessed. Then they tried to understand the given code and answered the comprehension questions. The results and comments of these subjects were used to refine the code examples and to arrange a sequence from easy to difficult.

In the feasibility study subjects first had to fill in a questionnaire about their programming experience. A differential analysis stratified in novices and experts is therefore possible.

In order to compare the reading of natural language text and source code, three short texts with comprehension questions were selected. Therefore the normal reading behavior of the subjects is contained and dyslexics etc. can be detected.

The first natural text and the first source code were used as dummies to familiarize the subjects with the task and the course of action, resulting in two NTs and ten SCs. The texts were formatted according to readability suggestions for text on screen, to ensure undisturbed reading.

We used a Tobii T120 eye tracker with Tobii Studio 2.0.6. The eye tracking equipment is integrated into a 17" TFT-Screen. This remote system is non-intrusive and does not need any further devices, which can be seen or heard by the subject. The sampling rate of 120 Hz was very high and relatively large movements of the head are possible. The study was conducted in a regular office setting. Working with a computer screen is a quite natural setting for programming task, so a very lifelike situation was achieved. During the session we calibrated three times to ensure good data quality. The test subjects first read the natural language texts, then the program texts. Afterwards they were shown their recording and commented on them (RTA).

A total of 15 subjects participated in the study. Their programming skills ranged from complete novice to experts in different languages.

3.2 Procedure of Analysis

Tobii Studio did not provide a sufficient analysis module, therefore a series of evaluation software had to be developed. Merely time, coordinates and duration of fixations were exported from Tobii Studio and processed with Matlab 7.4. In order to map the fixations to a word, the coordinates of the word have to be known, but manual capture of over 1000 words is barely feasible. So a tool was implemented using ImageJ, which identified the word's coordinates from an image containing the word's location on the screen and the stimulus text.

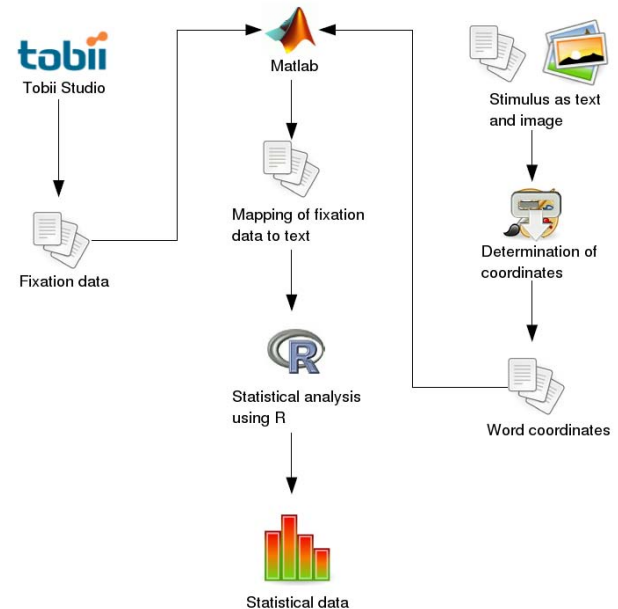


Figure 6 Workflow

Fixation data was mapped to the stimulus texts in Matlab. Every data set was examined for data quality. One subject had to be excluded, since only half of the samples were existent, due to technical issues. For another subject the samples for NT and two SCs were missing. The remaining 164 samples were plotted, manually inspected and corrected if needed: For some subjects, movements resulted in a systematic bias, so the eye data had to be adjusted. Due to the text structure most biased fixations could be mapped. Every intervention was documented in the data sheets.

3.3 Results

All continuous measures were tested for a Gaussian distribution but showed significant skewness, accordingly non-parametric tests were applied. Central tendencies and variability are described as median and 25th/75th percentile. Mean differences between two groups were tested with a Wilcoxon-test, multiple comparisons were tested by Kruskal-Wallis-tests. All analyses were conducted in R [19].

After mapping raw data to code elements in the respective stimuli, data sets with 512 to 1802 variables were available for further statistical evaluation, providing ten different types of information per element. One of the tasks of this feasibility study was the development of automated data transformation and analysis, including descriptive statistics, data visualization

and inference statistics. Only a few selected results will be presented here to illustrate the power of the eye tracking approach as well as its still existing challenges.

3.4 Mean fixation times for NT and SC

Before entering the area of complex reading patterns, we analyzed some very basic features of the reading process like mean fixation, i.e. the average fixation time of a code element. The global mean fixation accordingly is the median of these fixation times of all areas of interest.

For the two NT presentations, we observed a global mean fixation time of 258 (232 / 285) ms, well in accordance with the usual range of 200 to 400 ms found in literature. Averaging all ten SCs, mean fixation was 351 (309 / 408) ms (Figure 7).

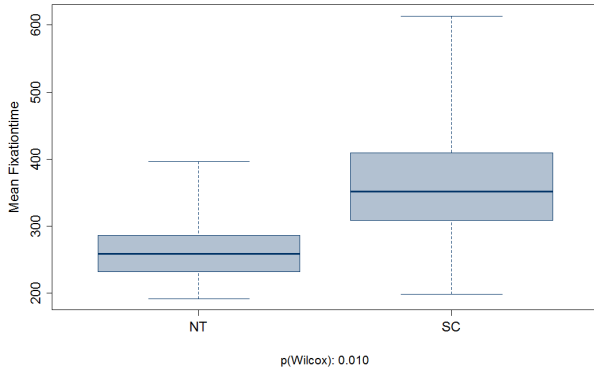


Figure 7 Global mean fixation time for NT (left) and SC (right)

The significant prolongation compared to NT indicates a substantial increase in demands in terms of attentiveness. For fixation time during first-pass as well as first-read we observed significantly higher values for SC compared to NT, confirming the finding in global fixation (data not shown).

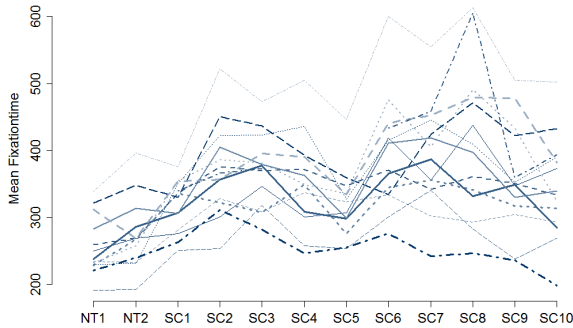


Figure 8 Mean fixation times of all participants

While all subjects showed this clear distinctive difference between NT and SC, there was quite substantial variability between subjects as well as between texts within the same category, which is depicted exemplarily in Figure 8 and Figure 9 by means of each participant's average fixation time over all texts and the mean fixation times per text.

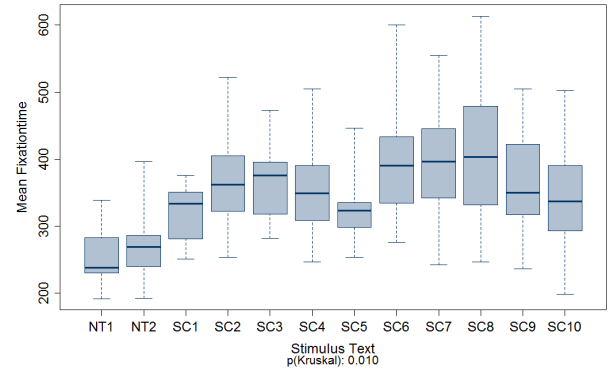


Figure 9 Mean fixation times of all texts

3.5 Number of regression

The number of regressions is another basic measure obtainable from eye tracking data.

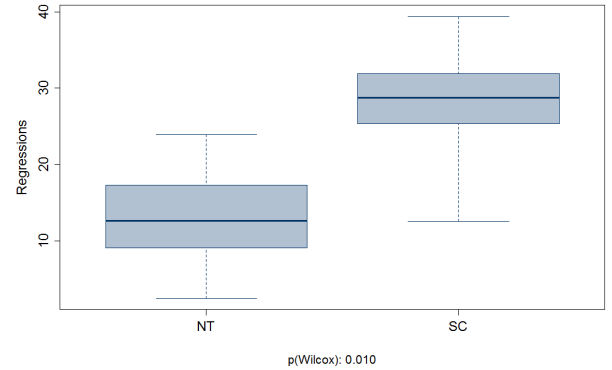


Figure 10 Regression rates for NT (left) and SC (right)

For NT, a rate of 10 to 15% has been reported previously, in our experiment we observed a rate of 16 (10 / 20)%. For SC, regressions were substantially more frequent with 37 (33 / 41)% (Figure 10). This significant increase indicates differences in complexity and information content between the two different types of text. Figure 11 clearly shows that SC made all subjects use extensively more regressive eye movements.

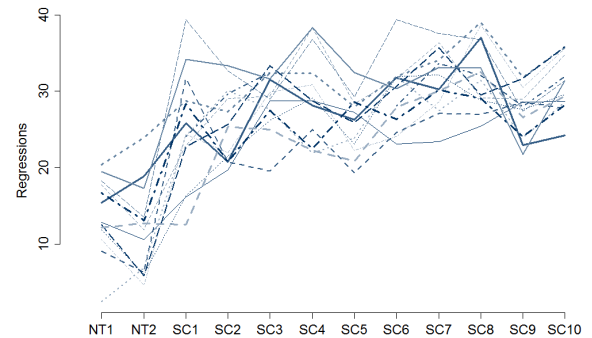


Figure 11 Regression rates of all participants

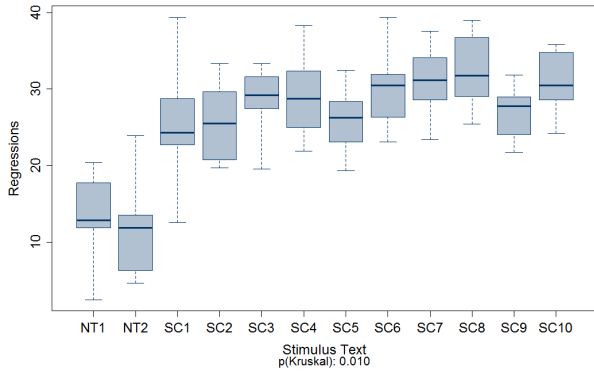


Figure 12 Regression rates in all texts

3.6 Categories of words within source code

The general distinction between NT and SC based on eye tracking is a prerequisite for attempting to apply this method to the differentiation between distinct types of words within SC. We defined keywords, identifier, numbers, and operators as our four categories of interest.

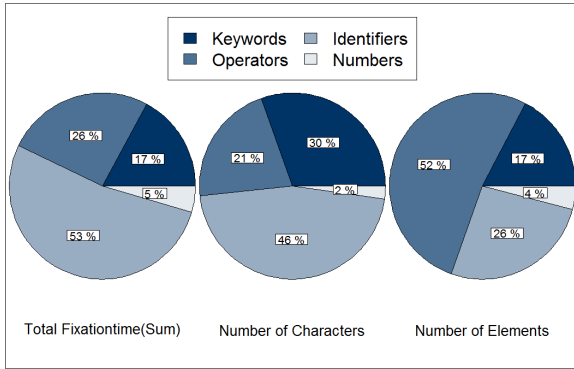


Figure 13 Fixation times for the different categories of words in SC (left), in comparison to the number of characters (middle) and the number of elements (right) for the respective categories

As can be seen in Figure 13 (left panel), the aggregated fixation time for elements of the type identifier is clearly the major fraction of total reading time, followed by operators and keywords. Reading of numbers took only a small fraction of time. Interpretation of this result is rather difficult, as aggregated fixation time may be influenced by the number of such elements as well as their length. Figure 13 (middle panel) depicts the number of characters for these categories, while the right panel shows the number of elements of each class.

The class of identifiers has the biggest number of characters, corresponding to the longest fixation time. For keywords and operators, there is no such correspondence but the order is reversed. Looking at the number of elements, identifiers make up approximately a quarter of all elements but take up about 50% of reading time. Operators on the other hand, while taking up a quarter of total fixation time, make up half of all elements of the SC tested. It becomes obvious, that there is no simple relation between the number of elements and the number of characters in a given category, nor is fixation time a linear function of either number. Accordingly, only within category comparisons are

appropriate between either readers with different level of knowledge (beginners vs. experts) or between SCs of varying level of complexity, where number and length of members of a word category can be kept comparable. Future analyses may either attempt to divide fixation time by syllables or units that can be processed with a single fixation in combination with statistical tools capturing the relation between word length and fixation within classes.

4. CONCLUSION AND FUTURE WORK

Within this pilot application of eye tracking technology in the study of perceptual processes in source code reading we were exploring possibilities as well as limitations of this methodology in furthering our understanding of program comprehension. At the current stage of analysis we already gained much confidence in the potential of this approach.

Our feasibility study led to the development of complex analytical tools for data processing from raw data to statistical inference. Several parts of this process could be automated by programming, leaving merely optimization tasks open. Some parts however, like artifact detection and correction, still have to be transformed from tedious manual work to some degree of automation.

For some of the measures obtainable from eye tracking like e.g. fixation time, more conceptual work has to be done in terms of adjustment to number and length of elements. Neither number of letters nor number of elements are suitable, as neither seems to reflect the true information content of elements.

Given the complexity of its methodology, eye tracking is not yet an ‘out-of-the-box’ method ready for widespread application, but it has itself established as valuable tool for specialized research groups. A glimpse of the potential may be possible by looking at the heat map generated from fixation times (Figure 14).

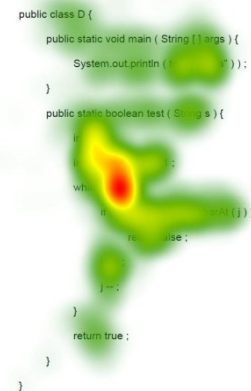


Figure 14 Heat map (by Tobii Studio)

The conventional heat map conceals the areas of highest interest, whereas the color coding (Figure 15) allows to still perceive the whole text. After all pre-processing of data, results show a clear hierarchy of areas of higher and lower interest, as depicted by the different color. Together with the order of fixations, this forms an ideal basis for discussing mental processes during the task of code evaluation.



Figure 15 Color coded attention allocation map (number of fixations per word)

As there is no sequential intake of SC line by line but rather a non-linear reading pattern driven by mental processes, reading becomes more of a creative task, actively driven by the programmer. Such observational data will add an empirical underpinning to theoretical concepts of program comprehension.

Based on eye tracking data, comparisons between subjects with varying level or styles of programming become feasible. An empirical evaluation of programming tools and IDEs would be possible, as well evaluation of teaching approaches, all based on this emerging new technique. Furthermore it can be used for the necessary corroboration of research findings, replicating classic experiments e.g. [15, 21].

A next step is to differentiate the results according to the subjects' level of programming expertise as elevated from the survey in order to gain characteristics of novice and expert programmers, like strategies. A possible use of those findings is to explicitly teach strategies that prove successful for certain tasks, like the scanning pattern of Uwano et al. for debugging (section 2.2). The effectiveness of such interventions can be reviewed again using eye tracking.

The other essential stage is to analyze the RTA data in order to deploy the potential of this study design. The consolidation of visual data and think aloud data is especially required, since participants tend to have a different visual behavior and attention focus than they assume.

Think aloud is already a valuable method in PC research. Its refinement in form of the combination of eye tracking and RTA overcomes limitations and provides a diversity of information, making it a more objective instrument to understand PC. This design can be adapted for a variety of research questions, like the analysis of programming knowledge and mental representations of specific programs.

5. REFERENCES

- [1] Aschwanden, C. and Crosby, M. 2006. Code Scanning Patterns in Program Comprehension. Symposium on Skilled Human-Intelligent Agent Performance. Measurement, Application and Symbiosis. Hawaii International Conference on Systems Science (2006).
- [2] Bednarik, R. and Tukiainen, M. 2006. An eye-tracking methodology for characterizing program comprehension processes. Proceedings of the 2006 symposium on Eye tracking research & applications (San Diego, California, 2006), 125-132.
- [3] Bednarik, R. and Tukiainen, M. 2005. Effects of display blurring on the behavior of novices and experts during program debugging. CHI '05 extended abstracts on Human factors in computing systems (Portland, OR, USA, 2005), 1204-1207.
- [4] Bednarik, R. and Tukiainen, M. 2008. Temporal eye-tracking data: evolution of debugging strategies with multiple representations. Proceedings of the 2008 symposium on Eye tracking research & applications (Savannah, Georgia, 2008), 99-102.
- [5] Bednarik, R. and Tukiainen, M. 2004. Visual attention tracking during program debugging. Proceedings of the third Nordic conference on Human-computer interaction (Tampere, Finland, 2004), 331-334.
- [6] Bente, G. 2004. Erfassung und Analyse des Blickverhaltens. Lehrbuch der Medienpsychologie. R. Mangold, P. Vorderer, and G. Bente, eds. Hogrefe Verlag für Psychologie. 297-324.
- [7] Block, A. 2002. Die Blickregistrierung als psychophysiologische Untersuchungsmethode: Grundlagen, Anwendung und technische Realisierung. Verlag Dr. Kovač.
- [8] Carreiras, M. 2004. On the On-Line Study of Language Comprehension. The On-line Study of Sentence Comprehension: Eyetracking, ERPs and Beyond. M. Carreiras, C. Clifton, Jr., and C. Clifton, Jr., eds. Psychology Press. 1-14.
- [9] Christmann, U. 2004. Lesen. Lehrbuch der Medienpsychologie. R. Mangold, P. Vorderer, and G. Bente, eds. Hogrefe Verlag für Psychologie. 419-442.
- [10] Crosby, M.E. and Stelovsky, J. 1990. How do we read algorithms? A case study. Computer. 23, 1 (1990), 24-35.
- [11] Crosby, M.E., Scholtz, J. and Wiedenbeck, S. 2002. The Roles Beacons Play in Comprehension for Novice and Expert Programmers. 14th Workshop of the Psychology of Programming Interest Group (2002), 58-73.
- [12] Galley, N. 2001. Physiologische Grundlagen, Meßmethoden und Indikatorfunktion der okulomotorischen Aktivität. Grundlagen und Methoden der Psychophysiologie. F. Rösler, ed. Hogrefe Verlag für Psychologie. 237-316.
- [13] Guéhéneuc, Y.-G. 2006. TAUPE: Towards Understanding Program Comprehension. Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research (Toronto, Ontario, Canada, 2006), 1-13.
- [14] Kerkau, F. 2009. Usability-Testing zur Qualitätssicherung von Online-Lernangeboten. Online-Lernen: Handbuch für Wissenschaft und Praxis. L.J. Issing and P. Klimsa, eds. Oldenbourg Verlag. 329-337.
- [15] Letovsky, S. 1987. Cognitive Processes in Program Comprehension. Journal of Systems and Software. 7, 4 (Dec. 1987), 325-339.
- [16] Lister, R., Clear, T., Simon, Bouvier, D.J., Carter, P., Eckerdal, A., Jacková, J., Lopez, M., McCartney, R., Robbins, P., Seppälä, O. and Thompson, E. 2010. Naturally occurring data as research instrument: analyzing examination responses to study the novice programmer. SIGCSE Bull. 41, 4 (2010), 156-173.
- [17] Lopez, M., Whalley, J., Robbins, P. and Lister, R. 2008. Relationships between reading, tracing and writing skills in introductory programming. Proceeding of the Fourth international Workshop on Computing Education Research (Sydney, Australia, 2008), 101-112.

- [18] von Mayrhauser, A. and Vans, A.M. 1994. Program Understanding - A Survey. Colorado State University Computer Science Technical Report CS-94-120. (1994).
- [19] R Development Core Team 2011. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing.
- [20] Rayner, K. and Pollatsek, A. 1989. The Psychology of Reading. Prentice Hall.
- [21] Soloway, E. and Ehrlich, K. 1984. Empirical studies of programming knowledge. IEEE Transactions on Software Engineering. 10, 5 (1984), 595–609.
- [22] TIOBE Software: Tiobe Index: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>. Accessed: 2011-06-30.
- [23] Uwano, H., Nakamura, M., Monden, A. and Matsumoto, K.-ichi 2006. Analyzing Individual Performance of Source Code Review Using Reviewers' Eye Movement. Proceedings of the 2006 symposium on Eye tracking research & applications (2006), 133-140.
- [24] Uwano, H., Nakamura, M., Monden, A. and Matsumoto, K.-ichi 2007. Exploiting Eye Movements for Evaluating Reviewer's Performance in Software Review. IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences E Series A. 90, 10 (2007), 317-328.
- [25] Venables, A., Tan, G. and Lister, R. 2009. A closer look at tracing, explaining and code writing skills in the novice programmer. Proceedings of the fifth international workshop on Computing education research workshop (Berkeley, CA, USA, 2009), 117-128.
- [26] Wittmann, M. and Pöppel, E. 2001. Neurobiologie des Lesens. Handbuch Lesen. B. Franzmann, K. Hasemann, D. Löffler, and E. Schön, eds. Schneider Verlag Hohengehren. 224-239.
- [27] Yusuf, S., Kagdi, H. and Maletic, J.I. 2007. Assessing the Comprehension of UML Class Diagrams via Eye Tracking. Proceedings of the 15th IEEE International Conference on Program Comprehension (2007), 113-122.