Developing Coding Schemes for Program Comprehension using Eye Movements

Teresa Busjahn Carsten Schulte Edna Kropp

Department of Computer Science
Freie Universität Berlin
{teresa.busjahn, carsten.schulte, edna.kropp}@fu-berlin.de

Keywords: POP-II.B. program comprehension, POP-III.B. java, POP-V.B. eye tracking, POP-VI.F. exploratory

Abstract

This paper introduces an approach to use eye movement data in the context of program comprehension studies. The central aspect is the development of coding schemes, which reflect cognitive processes behind the observable visual behavior of programmers. For this purpose, we discuss to first use a quantitative approach to find those episodes in the eye movements that yield the most potential for analysis. Subsequently, qualitative methods can be used on this subset.

1. Introduction

Tracking a programmer's gaze is probably one of the closest and most direct measurements we have to infer cognitive processes from behavioral and observable data. However, using eye movement data poses several challenges. Even though eye movements and cognitive processes are connected, the relation is complex and there is no easy matching. Moreover, eye tracking produces huge data sets. In this paper we discuss a structured approach to develop an analytical research instrument to study code reading and comprehension. Central for this approach is the development of coding schemes to label and aggregate eye movement data of programmers understanding source code.

As a start we will focus on eliciting program comprehension (PC) strategies from eye movement data. However, the presented approach is suitable for a multitude of questions, e.g. on the difference between novices and experts, the interaction of task type and comprehension process, influence factors like programming language and paradigm, program length, additional visualization, tools/interface issues, plan-like and un-planlike programs, and debugging.

The question we address here is "What strategies do expert programmers use during program comprehension?". We decided to look into experts first, since experts are supposed to have developed successful strategies which can be taught to less experienced programmers. Furthermore, we expect to find some established strategies that are shared by more than one individual.

In November 2013 the first international workshop on "Eye Movements in Programming Education: Analyzing the expert's gaze" was conducted as an attempt to broaden the knowledge about PC strategies. The focus was on cognitive processes behind observable eye movements during source code reading. The workshop was organized in association with the 13th KOLI CALLING Conference in Computing Education. Before the workshop, two sets of eye movement records of expert programmers reading Java were given to the participants.[1]

Participants were asked to analyze and code these records with a provided scheme containing code areas in different levels of detail, eye movement patterns and presumed comprehension strategies. Based on this analysis, the participants wrote position papers describing the eye movement data, commenting on the coding scheme, and possible applications of eye movement research in computer science education. The scheme was revised following suggestions given in the position papers and during the workshop.[2]

1 The data can be downloaded from http://www.mi.fu-berlin.de/en/inf/groups/ag-ddi/Gaze_Workshop/koli_ws_material.

In the following we will introduce a systematic approach to develop coding schemes for eye movement records of programmers in order to study PC processes - without getting lost in the data.

2. Using gaze data

2.1. Eye tracking in Program Comprehension

Eye tracking studies are offering a promising source of data for studying cognitive processes of programmers by 'showing' what is happening, without having to force the subject to think aloud. A problem with methods like think aloud is that they add an extra cognitive load besides the task at hand and therefore affect the comprehension process. Moreover, lots of program elements are never mentioned during a think aloud, even though they are important and taken into account by the programmer. This leads to an incomplete analysis of the mental state. In contrast, eye tracking provides the information which part of the program the subject was perceiving at exactly what point during understanding.

However, on a closer look some obstacles are occurring: The amount of data is bigger and more fine grained compared to e.g. verbal protocols. And also, it is not the thinking process itself that is made visible, but the process of reading something (in our case a program).

In the past, starting empirical studies in a new domain was often done using qualitative methods like Grounded Theory. While this approach seems useful in general, the complexity of eye movement data adds further challenges. In this approach, we suggest to take into account experiences from former eye tracking and PC studies to align the analysis process with the design of a central concept within this process, the coding scheme.

2.2. Eye Movements and Parameters

During reading, the eye stays on one point for a moment and then quickly jumps to the next location. These movements are called *saccades*, the relatively steady state between them *fixations*. When reading English, fixations usually last about 200-250 ms, but the duration can vary from under 50 ms to over 500 ms. The mean saccade length is 7-9 letter spaces, but can cover 1 to over 15 letter spaces. Saccades against the normal reading direction are called *regressions*. Typically, 10 - 15 % of saccades during reading are regressive. Good readers are characterized by short fixations and few regressions. However, these parameters depend on several factors like text difficulty and formatting. More demanding texts induce longer fixations, short saccades and frequent regressions (Rayner 1998).

3. Coding Schemes related to Program Comprehension

In qualitative research a code "is most often a word or short phrase that symbolically assigns a summative, salient, essence-capturing, and\or evocative attribute for a portion of language-based or visual data" (Saldaña 2012), p. 3. This data can have various forms, e.g. think-aloud transcripts, and video. Codes will most likely be used several times and form patterns. While coding, the data is organized and grouped into interrelated categories, which usually undergo refinements into different levels of subcategories. A further step is to compare and consolidate the categories to contribute to theory (Saldaña 2012). A coding scheme is an instrument that contains the possible codes and organizes them into categories.

3.1. A flexible expandable Coding Scheme for Software Comprehension

Von Mayrhauser and Lang (1999) describe a flexible expandable coding scheme (AFECS) to support the systematic analysis of PC. It bases on the integrated comprehension model[3] and is supposed to be consistent with accepted theories of PC. It was developed for protocol analysis on transcribed think aloud protocols reflecting programmer behavior.

2 The position papers and the full version of the coding scheme can be found in the technical report (Bednarik, Busjahn, & Schulte 2014) and at http://www.mi.fu-berlin.de/en/inf/groups/ag-ddi/Gaze_Workshop/.

3 See (Mayrhauser & Vans 1994) for a detailed description

The codes are systematically split into a number of segments, that encode particular aspects of a cognitive process. Starting point is the *mental model* ('program model', 'situation model', or 'domain model'). It reflects the level of abstraction at which a programmer is working. Programmers can start building a mental model at any level that appears opportune and switch between any of the three model components during comprehension. The second segment, *element* classifies what the programmer does at that level with the general notions of cognition 'goals', 'hypotheses', and 'actions that support the hypothesis driven understanding process'. These actions can be analyzed further and coded in segments with greater detail. Only the first two parts of the coding scheme (mental model and element) are mandatory.

The analysis proceeds from identifying actions of various types to determining action sequences and extracting cognition processes and strategies. The results can be used for statistical analyses, e.g. discovering patterns of cognitive behavior or analyzing frequencies of certain actions.

The coding scheme can be expanded or reduced according to the level of detail desired. Due to this flexibility, the scheme can be adjusted to answer a variety of research questions for various aspects of PC. Hence, researchers can tailor AFECS to their own needs instead of developing a coding scheme from scratch. Often results from different studies are difficult to compare. By using the same scheme, results maintain a degree of standardization and enable comparisons across studies.

This scheme is especially interesting in the context of our approach, as it directly provides a broad range of codes for cognitive processes during program comprehension.

3.2. An open-source Analysis Scheme for Identifying Software Comprehension Processes

O'Brien, Shaft, and Buckley (2001) compose an open analysis scheme for think-aloud protocols to determine the type of comprehension process used by programmers. This scheme extends the AFECS. It distinguishes between bottom-up and two variants of top-down comprehension. The first top-down type, *expectation-based* comprehension, goes back to Brooks. The programmer has a pre-generated hypothesis about the code's meaning and then scans it for that hypothesis. The second type *inference-based* comprehension is based on Soloway. The programmer scans the code, and derives an hypothesis from the incomplete knowledge about that program. The hypothesis is then validated against the code. Bottom-up processing is described as an initial study of code, line by line, leading to a general understanding of the program.

This scheme fits into the AFECS framework, giving more detail to the segment called *hypothesis action*. If the hypothesis action is generating a hypothesis, then the new segment describes the trigger for this generation process. Being open, the scheme allows for subsequent refinement by other researchers and for replication of experiments. Furthermore, it includes the analysis procedure used to assign verbal data to the elaborated categories.

3.3. A Scheme for Analysing Descriptions of Programs

Good and Brna (2004) present a coding scheme for analyzing free-form program summaries, which allow programmers to express their understanding in their own words at their chosen level of abstraction, including as much detail as they feel is necessary. Pennington's analysis schemes (Pennington 1987) was the starting point for developing this scheme. There are two kinds of classifications employed, *information types* and *object descriptions*.

The information types classification developed by Good and Brna contains 11 categories, e.g.

- function: the overall aim of the program, described succinctly
- actions: events occurring in the program which are described at a lower level than function, but at a higher level than operations
- control: information having to do with program control structures and with sequencing, e.g. recursion.

The object classification comprises how objects present in the program are described. There are seven object categories, e.g.

- program only: refers to items which occur only in the program domain, and which would not have a meaning in another context, like a counter
- program-domain: object descriptions which contain a mixture of program and problem domain references, e.g. a list of marks
- domain: an object which is described in domain terms, rather than by its representation within the program, e.g. a distance.

Possible analyses with this scheme are the proportion of information types used, and the level of abstraction featured in the summary. Good and Brna assume, that the scheme can also be applied on verbal protocols gathered during comprehension tasks.

3.4. The Base Layer

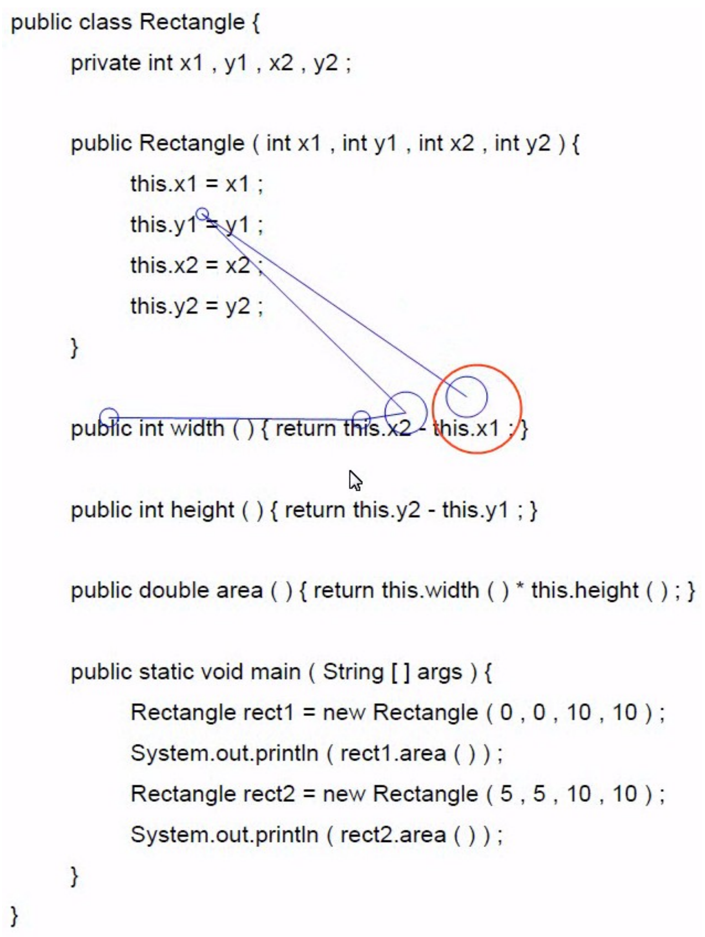
Based on Salinger (2013), Salinger and Prechelt suggest the idea of a base layer in context of studies on pair programming and the grounded theory methodology. The base layer consists of a set of predefined codes (the so-called base concepts), rules for changing these base concepts, including a naming scheme, and a general structure of the concept set. It aims to support a researcher to make faster progress, and to enable studies to be compatible so that results can be related to each other (Salinger & Prechelt 2013). Subsequent studies should be faster because they can use the given set of base concepts to create "higher-level concepts and eventually theory" (Salinger & Prechelt 2013), p. 28.

The idea is to give a kind of head-start, allowing a researcher to begin at a higher conceptual level while still being close to the data. Although this seems to introduce the danger of forcing the researcher to code theory-driven instead of data-driven, the authors claim to have taken some precautions against this danger: the set of base concepts is considerably small; allowing and inviting for additions and changes of the existing base concepts. The base concepts are explicitly not to be misunderstood as a coding scheme, but as a tool "to maximize the reader's capability of thinking flexibly about what it is that appears to be going on in the pair programming session and what might be an appropriate manner of conceptualizing it" (Salinger & Prechelt 2013), p. 35.

In addition, the base concepts aim to be neutral, generic and flexible, and not geared towards a specific research question. However, they are based on some theoretical assumptions, like speech act theory, due to the nature of the data (protocols of pair programmers' utterances during pair programming sessions).

4. Our current Coding Scheme

The previous schemes were mainly created for text data, either of think aloud protocols (von Mayrhauser & Lang and O'Brien, Shaft & Buckley) or of program summaries (Good & Brna). Only Salinger & Prechelt additionally considered audio and video recordings of pair programmers as well as screen castings. We will introduce a coding scheme that operates on eye movements of programmers understanding source code. The first version of the scheme was developed before the Koli Calling workshop. It is based on a short Java program defining rectangles (code 1) and two sets of gaze records by professional programmers. A fundamental decision was to distinguish between observable behavior and its interpretation and to classify codes accordingly.



```
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 ( ) ) ;
        }
}
```

Code 1 - Source code example used for the workshop (overlaid with eye movements)

Besides primitive categories that denote fixations on a certain point in the program, there are two categories of codes for a series of eye movements called *pattern* and *strategy*. Patterns are observable sequences of fixations, while strategies require the interpretation of a pattern concerning the intention behind this visual behavior. Several researchers involved in computer science education and eye movements defined an initial set of codes, observables as well as potential strategies. Only very few codes, like the scan pattern were adopted from previous research on eye movements in programming. This scheme was given to the workshop participants with the task to code the provided eye movement records using the video annotating software ELAN[4] and to modify the coding scheme as they seem fit.

The workshop participants' suggestions for the scheme were compiled into a revised version which was discussed during the workshop. Further revisions were included accordingly. Finally the observable codes of the scheme which only relate to single fixations were abstracted. Table 1 presents an excerpt from the final workshop coding scheme, table 2 a subblock example.

4 See http://tla.mpi.nl/tools/tla-tools/elan/.

Category	Codes	Description	Classification
(Lexical) Element	Public1 , Methodname1 , }1 . . .	(Lexical) element on which the fixation occurs, e.g. an operator or identifier	Observable
Block	Attributes, Constructor, Main, MethodX . . .	General area in which the fixation occurs, e.g. the main-method	Observable
SubBlock1, SubBlock2 . . .	MethodBody, ReturnLine, Signature, WhileHead, WhileBody . . .	Specific region in which fixation occurs, e.g. a signature or a line containing a return-statement. Can be nested. Granularity depends on structures of interest.	Observable
Pattern	Flicking, JumpControl, LinearHorizontal, LinearVertical, RetraceDeclaration, Scan, Word(Pattern)-Matching	Flicking: The gaze moves back and forth between two related items, such as the formal and actual parameter lists of a method call. JumpControl: Subject jumps to the next line according to execution order. LinearHorizontal: Subject reads a whole line either from from left to right or right to left, all elements in rather equally distributed time. LinearVertical: Subject follows text line by line, for at least three lines, no matter of program flow, no distinction between signature and body. RetraceDeclaration: Often-recurring jumps between places where a variable is used and where it had been declared (Uwano, Nakamura, Monden, & Matsumoto 2006). Form of Flicking. Scan: Subject first reads all lines of the code from top to bottom briefly. A preliminary reading of the whole program, which occurs during the first 30 % of the review time (Uwano, Nakamura, Monden, & Matsumoto 2006). Word(Pattern)Matching: Simple visual pattern matching.	Observable
Strategy	AttentionToDetail, DataFlow, DesignAtOnce, FlowCycle, Interprocedural-ControlFlow, TestHypothesis, Wandering	AttentionToDetail: Readers are trying to comprehend a piece of code that is not believed to contain bugs. In most cases, there is a slowness to AttentionToDetail, but the subject could also be verifying a global property, such as that argument/ parameter types agree or that the semicolons are present in the right places. DataFlow: Following a single object in memory as its value changes through the program. Can also occur backwards through control flow in service of debugging and/or program execution comprehension.	Interpretation

	DesignAtOnce: LinearHorizontal or Scan, hardly any jumps back. The subject's intention is to understand the general or algorithmic idea, without having the need to go into details. Aiming at understanding by linear reading of the complete (needed) code. Can easily be confused with excessive demand/trial and error, might also include TestHypothesis on local levels. Captures high-level algorithmic thinking, thus, features rather large steps as the gaze sweeps over the text typically associated with Linear and Scan patterns. Suggests reading through part or all of the code in a linear manner, intending to acquire an overall understanding of it.
	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. The Flicking pattern might then suggest the simplest level of the FlowCycle strategy.
	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.
	TestHypothesis: Repetition of a pattern or gaze path. Occurs in connection with DesignAtOnce or ControlFlow. The subject's intention is to check for some details in understanding. Hints at some issue where either the person was distracted, or which is more difficult to comprehend. Involves repetition of a pattern of gaze, and suggests further concentration in order to better understand a particular detail.
	Wandering: It appears that the subject was backtracking, seemingly searching for a point to resume the reading after a particular path of reasoning had been exhausted, essentially a transition period or a brief rest between bursts of effort.

Table 1 - Workshop coding scheme (excerpt)

Using this scheme on eye movement records provided an excellent basis for the rich discussion during the workshop. The codes were developed partly top-down and partly bottom-up on only two eye movement records on a single program. It is not possible to draw conclusions about the reliability or the completeness of the scheme. Additionally, it is hard to describe the codes unambiguously, some codes are still rather fuzzy. These shortcomings lie somewhat in the nature of the data, the analysis instrument and the kind of research problem. Nevertheless, applying the scheme illustrated the usefulness of this kind of analysis and we can draw from the lessons learned.

Category	Codes	Description	Classification
Signature	FormalParameterList, Name, Type, Visibility	Precise code section on which a fixation occurs: the signature of a Java method	Observable

Table 2 - Example of a subblock

5. A systematic Approach to develop new Schemes

While the current coding scheme shows that using eye movement records have the potential to be used in PC research to study cognitive processes, a more systematic approach for developing such coding schemes seems needed, especially with regard to evaluating the scheme and coding reliability. A purely data-driven approach is not feasible. It takes circa two hours to code a one minute eye movement record and the the coding was experienced as being very tedious by the coders. Moreover, even though circa 10 coders worked on the data sets, it did not seem like some point of saturation of codes was reached.

Therefore we suggest to integrate qualitative and quantitative methods into a combined research design as suggested e.g. by Mayring (2001). Different models for this are possible. We opt for first applying a quantitative approach on the huge amount of eye movement data and use the results to decide, which data is suitable for qualitative analysis. Thereby we first reduce the data to make a qualitative analysis possible. The second analysis step is an interpretation of the eye movements that were identified as relevant, deepening the understanding.

Drawing on the idea of systematically distinguishing between observable visual behavior and inferences of cognitive processes, resulting coding schemes will have again the two different kinds of codes: observables and interpreted cognitive processes. In the following, we will discuss possibilities and alternatives for these two parts of a coding scheme.

5.1. Finding relevant Patterns in Eye Movement Data

Due to the vast amount of data, it is not feasible to analyze complete eye movement records for cognitive processes. Therefore we suggest to apply a quantitative approach to first find those patterns in the eye movements that occur most often to reduce data to be analyzed qualitatively.

A reasonable procedure is to first compute pairs of fixations that appear most often. Subsequently, the same will be done for sequences of three, four and more fixations. This way, jumps from specific individual elements to other elements can be examined. For this, each element in the program needs a unique identifier as indicated in the current scheme in category 'element'. This leads to the most frequent transitions from certain elements to others, which can be qualitatively analyzed to find the reasons for this often occurring behavior or to associate cognitive processes. But in itself, looking at specific elements is not very meaningful, especially when looking at different programs. Hence, more abstract types of elements have to be studied, like jumps from elements of one lexical category to the same category, and to the other categories. Moreover, switches between even broader types of program elements are of interest.

Sharma, Jermann, Nüssli, & Dillenbourg (2012) suggest to organize Java tokens into the three semantic classes identifiers (I), structural elements (S) and expressions (E).

- Identifier: variable declarations
- Structural: control statements
- Expression: main part of the program, like the assignments, equations, etc.

They regard 3-way transitions as one unit of program understanding behavior. It is suggested that a programmer switching between identifiers and expressions tries to understand the data flow and/or the relation among the variables. Transitions among all the semantic classes indicate the intention to understand the data flow according to the conditions in the program. Table 3 shows the categorization of different transitions among the semantic classes.

Type of flow in the program	Types of transitions
Data flow	I → E → I, E → I → E
Control flow	I → S → I, S → I → S
Data flow according to Control flow (Systematic execution of program)	S → E → S, E → S → E, S → I → E, E → I → S, S → E → I, I → S → E, I → E → S, E → S → I

Table 3 - Categorization of different transitions among semantic classes according to (Sharma, Jermann, Nüssli, & Dillenbourg 2012)

On a more general level, it is worth trying to find the most common 'global' patterns, how programmers go about understanding a source code. Finally, we'd like to choose a few control structures that are of special interest, e.g. loops and conditions. For these longer sets of fixations, instruments to compare fixation sequences as proposed e.g. by Cristino, Mathôt, Theeuwes & Gilchrist (2010) and West, Haake, Rozanski & Karn (2006) can be applied in addition to counting frequencies.

Besides computing possible data points to analyze, it is still a good idea to have a human looking for interesting data sequences. There might be patterns which are not frequent but nevertheless yield rich information, like extreme or very unexpected behavior or ideal cases that can be predicted from current theory.

5.2 Eliciting Cognitive Processes

A qualitative approach will be employed to explore cognitive processes. Instead of analyzing the whole gaze record, only those patterns identified in the quantitative step are looked at. Even after this reduction, plenty of data remains. This data is still in form of eye movement records. There are fixations on the program and saccades between them. This can be displayed as an animation, a video or in form of a graph representing the sequence and duration of fixations (see Code 1). As a start, it would be reasonable to find adequate names for the patterns.

As potential methods, we will discuss qualitative content analysis, phenomenography and grounded theory. They share a related initial analytical approach, in which phenomenography and grounded theory go beyond content analysis to develop theory or a distinctive understanding of the experience (Hsieh & Shannon 2005).

Although there are some other schemes we could draw potential codes from (see chapter 3), we will concentrate on a data-driven approach. The advantage is that results are gained directly from the data without imposing categories or theory. While codes in content analysis can be created either data-driven or derived from theory, in phenomenography and grounded theory categories emerge from within the data. In the following, these three approaches are introduced and their potential for the intended procedure is discussed.

5.2.1. Qualitative Content Analysis

Qualitative content analysis is "a research method for the subjective interpretation of the content of text data through the systematic classification process of coding and identifying themes or patterns" (Hsieh & Shannon 2005), p. 1278.

Qualitative content analysis focuses on texts within their context of communication, the data can be all kinds of recorded communication. There are different approaches, of which conventional, directed, and summative are used often.[5] These approaches differ among other things in the source of codes. Directed content analysis uses existing theory or research to derive the initial coding scheme before

5 Mayring (2000) refers to conventional content analysis as inductive and directed content analysis as deductive category development.

analyzing the data. It aims at extending or refining an existing theory. The summative approach counts single words or content and interprets the underlying context. Conventional content analysis is generally used to gain a richer understanding of a phenomenon, when prior theory or research is limited. The coding scheme is derived from data during data analysis. Codes are sorted into categories and relationships among categories are identified (Hsieh & Shannon 2005; Mayring 2000).

The overall intention of qualitative content analysis to interpret meaning from content matches our proposition. Furthermore the level of our intended outcome corresponds to what seems feasible with qualitative content analysis, producing a coding scheme with categories and codes describing PC processes in order to contribute to theory building. Nevertheless, eye movements are not exactly the kind of data, this approach aims to analyze.

5.2.2. Phenomenography

Phenomenography is an empirical, qualitative research approach that describes the variation in the way people understand or experience a certain phenomenon. The analysis consists of an iterative process, in which the researcher goes back to the data again and again. The outcome space of this analysis is a set of categories specifying different levels of understanding. These categories often have a hierarchical structure, going from categories with few features of the phenomenon to depicting richer or deeper understanding. Phenomenography is usually used in educational settings (Eckerdal 2009). The data for phenomenographic research has the form of people's accounts of their own experience and is usually gathered via interviews. Richardson (1999) points out, that other data sources are possible. Though those are in general just other forms of discourse that have the same evidential status as oral accounts.

While the goal to find ways in which programmers understand source code in general agrees with the phenomenographic paradigm, the intended outcome is still different. At the current point, it is of interest to develop a coding scheme to capture different cognitive processes during PC. The outcome space obtained by phenomenography is already a step further than what seems reasonable right now for the coding scheme.

5.2.3. Grounded Theory

While there are different versions of Grounded Theory Methodologies (GTM), they share some common features. Essentially, the idea is to generate theory from data, by repeatedly comparing and analyzing sections of data (open and intermediate coding), adding new data during the research process (theoretical sampling), until the more and more abstract codes and categories can be linked to a theory, explaining or describing the phenomena embedded in the data. Such a theory is conceptualized as emerging from the data.

An important characteristic is the intertwined process of ongoing analysis and generation of new data; connected to the cyclic approach to coding, where the codes and categories are constantly compared to new data and new codes. This nature allows the resulting theory to focus on those important characteristics that are in the data, not in some pre-defined research hypothesis. To allow the research to be open to the data, the role of literature and current state of research is somewhat ambivalent in GTM. When the methodology was first devised by Glaser and Strauss (1967), the "grounding" of theory directly in qualitative data was supposed to replace an uncritical acceptance of existing theory (Richardson 1999). The codes and categories emerging from the data might be very different from what would be expected from previous research. Theory or literature is used prior to research to sensitize the researcher, or during the coding process, when the theory emerges, as another perspective for comparing codes.

So far, GTM seems to be a suitable option for the data-driven analysis of visual behavior while understanding source code. However, it remains to discuss, what representation of the gaze data is needed to use GTM.

6. Conclusion

We presented an approach to use eye movement data for PC studies. For that purpose, we combine quantitative and qualitative methods to develop coding schemes. As an initial example, we worked on a scheme about comprehension strategies by expert programmers. Taking into account previous coding schemes in this context and the procedures of their development, allowed us to reflect potential pitfalls such as the missing comparability of results in advance.

Coding schemes for eye movement data should contain observable behavior as well as interpreted cognitive processes. For the most part, the observable codes can be assigned automatically, which is an advantage over previous coding schemes. Following the proposed procedure facilitates the comparison of data-driven results with other studies, without having to adopt their theoretical premises. Having a consistent, but yet flexible naming scheme as suggested by von Mayrhauser & Lang (1999) and Salinger & Prechelt (2013) will help that.

In order to use the above discussed qualitative methods, the gaze data could be translated into textual form using observable codes as 'label'. The resulting records would have this form: Signature - MethodBody - MethodBody or Scan - JumpControl - LinearHorizontal, enriched with information on the line and an unique name for the element. This might be seen as a representation of the raw data, similar to the transcript of an interview. However, unlike a transcript, any chosen label is already implying a certain interpretation. Hence, this translation process has to be done carefully. It is interesting to now explore the possibility to produce such a representation of the eye movements in a rigorous, and probably automated or semi-automated way.

7. Acknowledgements

We would like to thank Shahram Eivazi, Tersia //Gowases, Andrew Begel and Bonita Sharif as well as the other participants of the Koli workshop for discussing the ideas presented in this paper.

8. References

Bednarik, R., Busjahn, T., & Schulte, C. (2014). Eye Movements in Programming Education: Analyzing the expert's gaze. Joensuu, Finland: University of Eastern Finland.

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.

Eckerdal, A. (2009). Novice Programming Students' Learning of Concepts and Practise. Uppsala University, Uppsala.

Good, J., & Brna, P. (2004). Program comprehension and authentic measurement: a scheme for analysing descriptions of programs. Empirical Studies of Software Engineering, 61(2), 169–185.

Hsieh, H.-F., & Shannon, S. E. (2005). Three approaches to qualitative content analysis. Qualitative health research, 15(9), 1277–1288.

Mayring, P. (2000). Qualitative Content Analysis. Forum Qualitative Sozialforschung / Forum: Qualitative Social Research, 1(2).

Mayring, P. (2001). Combination and integration of qualitative and quantitative analysis. In Forum Qualitative Sozialforschung/Forum: Qualitative Social Research (2).

O'Brien, M. P., Shaft, T. M., & Buckley, J. (2001). An Open-Source Analysis Schema for Identifying Software Comprehension Processes. In Proceedings of 13th Workshop of the Psychology of Programming Interest Group (p. 129–146). Bournemouth, UK.

Pennington, N. (1987). Stimulus structures and mental representations in expert comprehension of computer programs. Cognitive Psychology, 19(3), 295–341.

Rayner, K. (1998). Eye Movements in Reading and Information Processing: 20 Years of Research. Psychological Bulletin, 124(3), 372–422.

Richardson, J. T. E. (1999). The Concepts and Methods of Phenomenographic Research. Review of Educational Research, 69(1), 53–82.

Saldaña, J. (2012). The coding manual for qualitative researchers. Sage.

Salinger, S. (2013). Ein Rahmenwerk für die qualitative Analyse der Paarprogrammierung. Freie Universität Berlin, Berlin.

Salinger, S., & Prechelt, L. (2013). Understanding Pair Programming: The Base Layer. BoD–Books on Demand.

Sharma, K., Jermann, P., Nüssli, M.-A., & Dillenbourg, P. (2012). Gaze Evidence for Different Activities in Program Understanding. In Proceedings of 24th Workshop of the Psychology of Programming Interest Group (p. 20–31). London, UK.

Uwano, H., Nakamura, M., Monden, A., & Matsumoto, K. (2006). Analyzing individual performance of source code review using reviewers' eye movement. In Proceedings of the 2006 symposium on Eye tracking research & applications (p. 133–140). San Diego, California: ACM.

Von Mayrhauser, A., & Lang, S. (1999). A coding scheme to support systematic analysis of software comprehension. Software Engineering, IEEE Transactions on, 25(4), 526–540.

Von Mayrhauser, A., & Vans, A. M. (1994). Program Understanding - A Survey. Colorado State University Computer Science Technical Report CS-94-120.

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 (p. 149–154). San Diego, California: ACM.