An Exploration of Shared Visual Attention in Collaborative Programming

Sami Pietinen, Roman Bednarik, Markku Tukiainen
Department of Computer Science and Statistics, University of Joensuu, Finland
{firstname.lastname}@cs.joensuu.

Keywords: POP-I.A. group dynamics, POP-II.C. working practices, POP-V.A. attention investment, POP-VI.F. exploratory

Abstract

This work-in-progress paper reports on the initial results of an eye-tracking research of collaborative program development, more particularly, in the case of pair programming. The study was conducted in nearly industrial-like settings and focused on pair programming productivity and improvement of the protocol that is followed by the pair-programming pair. Results are based on both the practical experience during the study and on an analysis of the recorded eye-movements. We present a descriptive analysis of visual attention during pair programming. We speculate that the pair-programming protocol is partly visible in both participants' eye-movements and can be used as an additional source of evidence when investigating the true protocol that the pair actually follows. We discuss the collaborative view on Psychology of Programming (PoP), which extends it to Psychology of Software Engineering (PoSE) and we then outline the future directions of our research.

1. Introduction

One of the important skills in collaborative work and learning is the ability to attend to information of mutual interest. In fact, humans are experts in coordination of our activities based on the information we get about the point of regard of our peers. Together with other social communication cues, such as facial expressions, gestures, or posture, we continuously monitor gaze direction of other collaborators, and we use these to contribute to our understanding of joint attention as we communicate and work with others. Because visual attention is an important skill in programming and in pair-programming, we began investigating its aspects in collaborative program development. In this paper we report on the initial results of this research.

Experimental methods have been used in most of the pair programming (PP) studies, but applying eye-tracking to study activities during collaborative programming has been rare to date. The present paper describes results concerning PP productivity from a research in which a complex environment was set up for two two-month-long software development projects. In the projects, two programmers worked as a pair within a larger, geographically distributed project team. The development was carried out using a single computer display. Pair Programming (PP) has been used for long time in various forms but gained the qualified name and general interest due the introduction of eXtreme Programming (XP) by Kent Beck. PoP research has followed this kind of development by extending its interests to group activities (Sajaniemi, 2008), which also covers PP. This paper is based on Pietinen, Tenhunen and Tukiainen (2009) containing further elaboration on the subject matter.

2. Related work

2.1 Pair Programming

Definition. Pair programming is a method where two persons work together with an algorithm, design or programming task using one computer (Williams and Kessler, 2000). According to the literature, PP is said to yield several benefits: code has less errors, designs are simpler, problem solving is more efficient, co-operation and communication increases, productivity is better, and integrating newcomers into teamwork is faster (Williams, Kessler, Cunningham and Jeffries, 2000; Succi and Marchesi, 2001; Williams and Kessler, 2003).

Benefits. Still, there have been only few empirical PP productivity studies, and their results have been controversial about the claimed benefits; in particular, the alleged better quality and productivity has

been questioned (Hulkko and Abrahamsson, 2005). On the other hand, Preston (2005) represents collective evidence for better quality in academic teaching context by referring to (DeClue, 2003; McDowell, Hanks and Werner, 2003; McDowell, Werner, Bullock and Fernald, 2003; Williams and Upchurch, 2001). The better quality is also supported in (Williams, Kessler, Cunningham and Jeffries, 2000) for both professional and student programmers. Better problem solving and learning is supported in (Williams, 2000). In overall, the use of PP seems to be more easily justified in the academic context than in the industrial context.

Task Dependency. However, PP does not benefit all software development tasks (Williams and Kessler, 2003). It is not clear, for example, what phases of software project, a single programmer, or a pair, or a team of programmers would better conduct development. It seems that the level of system complexity (Arisholm et al., 2007) and programmer expertise (Arisholm et al., 2002; Lui and Chan, 2006) have an effect to the productivity of PP. Pairing of experts is more productive with lower complexity systems in contrast to novice programmers. The correct personality match helps avoiding conflicts (Williams and Kessler, 2003). According to Williams and R. Kessler (2000b), PP is beneficial in design and code reviews, and tasks related to architecture and coding.

2.2 Pair Programming Productivity and Practice

Productivity of Software Development. The ultimate goal of improving software development methods is to increase productivity. This can happen either by lowering costs, increasing quality or size of output, shortening time to market, or a mix of these. Software productivity has become a critical problem primarily because the demand for new software increases faster than the industry's ability to supply it. Increased demand for software is the result of pressure throughout the economy to improve commercial, industrial, and service productivity (Boehm, 1981). With the recent push to downsize or outsource, software industry is trying to look for ways to cut down the software costs. Today, the majority of improvement strategies being pursued try to either reduce the inputs (i.e. people, time, equipment) needed to do the job, or increase the outputs generated per unit of input (Reifer, 1997).

While looking for ways to increase productivity, it is possible to not only improve the methods of programming, but also to improve other activities than just programming. According to Boehm (1981), all the factors influencing software productivity is not under our control, but one of the primary controllable factors is the number of instructions we choose not to develop, either by deferring the development of marginally useful features or using already available software like commercial software packages, adaptation of existing in-house software or using program generators.

Productivity of Pair Programming. In Pair Programming, no indications of the superior productivity of PP over solo programming could be detected based on a multiple case study (Hulkko and Abrahamsson, 2005). PP effort levels reported in various other studies have been found from -28 % to 100% (e.g. Arisholm et al., 2007; Baheti, Gehringer and Stotts, 2002; Ciolkowski and Schlemmer, 2002; Lui and Chan, 2006; McDowell et al., 2003; Mendes, Al-Fakhri and Luxton-Reilly, 2005; Rostaher and Hericko, 2002; Vanhanen and Lassenius, 2005; Williams, Kessler, Cunningham and Jeffries, 2000; Pietinen, Tenhunen and Tukiainen, 2008)[1].

In our case (Pietinen, Tenhunen and Tukiainen, 2008), the productivity of PP was lower compared to solo programmers. When the used software development process is more mature, i.e. uses standard reviews or inspections, than the one we used, difference of even 15% in productivity might be acceptable due to the additional review phase needed with solo programmers (Crosby, Scholtz and Wiedenbeck, 2002). But with our partial use of PP, an additional review phase would also be needed, so roughly less than 10 % would be acceptable and this threshold was overrun. The function point based productivity analysis still left some hope with comparable productivity, but the use of high-level program features in sizing might lead into results that are not inside the expected accuracy of the sizing method. There are still other benefits of using PP, like the increased learning, which might be

[1] The effort level is interpreted so that a percentage over zero means overhead in total efforts compared to solo programmer situation i.e. 100 percent means doubled effort.

considered highly relevant especially concerning tacit knowledge. But what comes to effort-based productivity, our results showed clearly a negative effect.

Quality Assurance Using Reviews. Reviews – published also as an IEEE Standard (IEEE Std. 1028-2008) - during the ongoing implementation surely contributes to the improvement of the product quality, as they are considered to be one of the most cost-effective ways to identify and remove defects in early stages of system life cycle (Parnas and Lawford, 2003), but many benefits come on the other hand from true collaboration and communication between the developers working as a pair, making them succeed in their tasks better than they could individually (Williams and Kessler, 2003). In Müller (2005), support is provided for reviews being an alternative to PP if same functional correctness is required from both pair and solo developers. This study used students as participants. He also found that PP pairs produced more correct solutions regards to number of program failures but with higher cost, although this was not found to be statistically significant.

Methods to Study Pair Programming Protocol. One productivity factor of PP is the parallelism of working i.e. doing multiple, also sometimes partly different, things at the same time, but also to the protocol i.e. the practice, which directs who is doing what and how it is coordinated between two people. There have been few studies on PP protocol in which video and audio (Höfer, 2008), dialog-based protocol, and Hierarchical Hidden Markov Models (Damiani and Gianini, 2007) where used for investigating what really happens during a PP session. According to Shaochun and Rajlich (2005), the recorded verbal protocol should be called as dialog based protocol (applied in e.g. Shaochun, Rajlich and Marcus, 2005) rather than think-aloud protocol, because this protocol can lead to more complete documented process as the dialog happens in more natural settings and there is continuous verbalization, and the method also reduces the Hawthorne effect. They do not argue, that the complete process gets documented, but rather say that this might be an improvement to the traditional think-aloud protocol, which they say referring to Ericsson and Simon (1993), can produce only a subset of thoughts. Few recent studies have questioned the underlying driver-navigator role division (Chong and Hurlbutt, 2007; Bryant, Romero and du Boulay, 2008). They discovered that the collaborative work happens to some extend in same abstract levels of thinking between pairs, not in completely separate levels. We propose eye-tracking as a complementary study method to the set of methods used in previous studies.

2.3 Visual Attention in Programming

Eye-Tracking. To better understand and eventually further improve the PP process we applied eye-movement tracking to capture the visual attention patterns of two programmers. The eye-tracking research rests on the "eye-mind assumption" (Just and Carpenter, 1980), a link between visual attention and visual information, assuming that attention is linked to foveal gaze direction. It however also acknowledges that it may not always be so (Duchowsky, 2003). We can focus our visual attention to a particular point and at the same time think something else. Also, we can sometimes use our peripheral system to direct our focus in the field of view instead of foveal system, the latter one being where the contemporary eye-trackers are basing their function on. An eye-tracker is a device that measures the gaze direction of a single subject and reports the location of the focus of subject's gaze. Previously, eye-tracking has been used in areas such as usability research, computer supported cooperative work, psychology of programming and medical science, just to mention few.

Visual Attention is a Skill. Because visual attention is an important skill in software development, numerous studies have applied eye-tracking to investigate the patterns of visual attention of programmers during various program maintenance tasks. For example, if an integrated development environment (IDE) presents different views on the program or project in several windows, programmers need to coordinate these representations (Romero et al, 2003). Other studies have investigated how the role of representations is changing as programmer builds the mental model and acquire more knowledge about the program (e.g. Bednarik and Tukiainen, 2007; Bednarik and Tukiainen, 2008; Bednarik et al., 2005b) and what are the effects of expertise on visual attention during programming activities (e.g. Crosby and Stelovsky, 1990).

Regarding the expertise effects on visual attention during solo programming, Crosby and Stelovsky (1990) provide no evidence for systematic differences between reading strategies of novices and experts programmers. They however observed that novices tend to concentrate upon comments while experts paid more attention to meaningful and complex areas in the code and rely on them. In a later study, those complex areas and statements were related to so-called beacons (Crosby, Scholtz and Wiedenbeck, 2002). Bednarik and Tukiainen (2007) found that experts exhibited more efforts to integrate the information available from different representations. Experts were also able to change the visual strategies to relate code and output information more at the later stages of debugging. Novice programmers, on the other hand, alternated between a limited set of strategies.

Joint-Attention in Collaborative Work. In collaborative situations, the eye-gaze is a central element of successful communication, and particular eye-movement behaviour help regulate many of the aspects of interaction with others. It is believed that joint-attention – i.e. the shared point of regard – is one of the underpinnings in early learning (Butterworth, 1995) and it allows us to better infer internal states of person we communicate with. For example, in collaborative problem solving, one of the requirements for successful ongoing process (e.g. for pair-programming) is efficient communication (Flor, 2006).

We believe that eye tracking can be used to study various aspects of PP. There are numerous interesting questions that are related to our general research interest: "what is the role of visual attention in pair programming activities?" As there are different views on how the PP protocol actually works, i.e. what actually happens when a pair is engaged in a programming activity, we choose to begin with this problem.

3. Method

3.1 The Research Project

A combination of agile development process together with pair programming (PP) is regarded as a cost-effective method (Succi and Marchesi, 2001). The current multinational software development often happens in geographically distributed environments. The data was collected during a research project called *Software Productivity* (SoPro), aimed at understanding the issues of productivity in software engineering. In the SoPro research project, we focused on enhancing PP to suit better for the geographically distributed project teams.

The project was carried out in three phases. First, an assessment of the software development processes of the participating companies was conducted; second, the project continued by identifying problems and needs in implementation phase; and third, when one of the possible method of a pair programming was developed and implemented. To this end, the method has been, and will further be, developed and piloted in real software projects.

3.2 Research Environment

We began this research project by setting up an eye-tracking system to record two programmers' visual attention at the same time. The system consists of two separate eye-tracking cameras for capturing the eye movements of both programmers. The data are stored separately at two computers and can later be analyzed in parallel. We are using one monitor-integrated eye tracker and one table-mounted eye-tracker from different tracker manufacturers but it is in our plans to test a single manufacturer setup for getting better comparability of eye movement data. The system is very prone to loosing the tracking of an eye with the table-mounted eye-tracker, so improvement in this regard is also needed. More elaborate description of the eye-tracker setup can be found from (Pietinen et al., 2008). We have also developed a tool that allows for analysis of multiple eye-tracking recordings.

To accurately measure and analyze various aspects of the development process and product, several protocols were recorded: an overview video with sound, video with sound showing the content of monitors screens, and the eye-tracking videos showing eye movements superimposed on top of the monitor screen video captures. Also, both developers kept a diary about ongoing activities. This paper

offers a first glance to the recorded eye tracking data and suggests ways to link it to improvement of PP practice.

4. Results

In the following, the underlying role division is considered as belonging to under the term pair programming protocol i.e. to the PP practice that is followed. At this point, we settle for descriptive exposition of visual attention in PP and suggest improvements to the underlying protocol, or practice, followed by the pair. As there is huge number of collected eye-tracking data, recorded during two two-month software development projects, we will extend the data analysis to numerical and statistical methods in the future to make it feasible in the first place.

4.1 Our Experience on the Pair Programming Practice

The importance of protocol. The efficiency of PP is very much dependant on the way how it is implemented and the tasks to which it is used. There are environmental requirements in PP such as a room, where one can discuss about ongoing task without feeling being disturbing other peoples' work and a table, which can accommodate two people side by side. There are also people requirements, because doing the work as pair work means fluent and nearly constant dialog and there might be conflicts and large gaps related to personality and experience differences. Still, these requirements seems in a way secondary, because even if the previous factors can be set to optimum level, there is a good change that the pair will not work as efficiently as possible, if the underlying protocol that they are following does not work well.

The division of roles. The traditional way of dividing the work to two different levels of thinking was done so that the programmer was seen to do more operational level if thinking by producing code or design and the navigator was continuously reviewing the results and thinking simultaneously the big picture in strategic level (Williams and Kessler, 2003). The traditional division on level of thinking – the division to operational and strategic thinking - between roles of a pair is only partly correct, or could be described as inaccurate. Although the underlying assumption in division of level of thinking has been questioned, the implications of this to the PP protocol have been very little presented.

Avoiding free driving. Almost all of the driver's attention goes to writing code and doing just quick designs when needed. The navigator on the other hand is left with spare time from the review task, because you can definitely review the code faster then one can write it. Sometimes you just know what the pair is doing, you check the code quickly for bugs and other issues, you know you are going in the right path and not so much communication and attention is needed. These are the points, where there is conscious choice to be made. Either you don't do anything the fill the spare time (free-driving, introduced in e.g. Williams and Kessler, 2003) or you make up something to do. If you already know what's the next small sub-task to do, which should be mostly the case, maybe one can start drawing some designs or test cases to it. Without acknowledging these situations, there is a threat that the navigator can get bored and frustrated because of lack of engagement and focus. Changing the roles e.g. every fifteen minutes helps getting engaged more often, which is important, but for being productive as possible the spare time needs to be filled with something useful to the shared goal.

Information search. Sometimes a great amount of time goes to searching needed information e.g. what third-party component are available or examples on how to use some API-call. In simple information search tasks that still require, lets say more than 10 minutes to complete, pairing can be a total waste of time and for example SbSP model might be preferred. For these situations, there should be opportunity to use another computer and review and merge the search results once in a while to prevent doing things twice. Using efficient sub-tasking, merging of results could happen just even just once as a post-procedure. SbSP requires two workspaces, but as a word of warning, the opportunity to use another computer can lead into decreased use of PP and can therefore make things worse if one is trying to achieve full-time use of PP and avoid discontinues and partial use of PP.

4.2 Visual Attention (VA) in Pair Programming

Description of VA in PP. Figure 1 shows what the resulting eye-movement data looked like during a coding task when viewing it retrospectively as superimposed on top of screen capture. On top, there is driver's eye-gaze representation and at the bottom the navigator's. The coloured circles (blue and canyon) represent the current Point-Of-Gaze (POG). The blue and red lines ending to POGs are gaze trails that show the POG history for the length of few seconds.

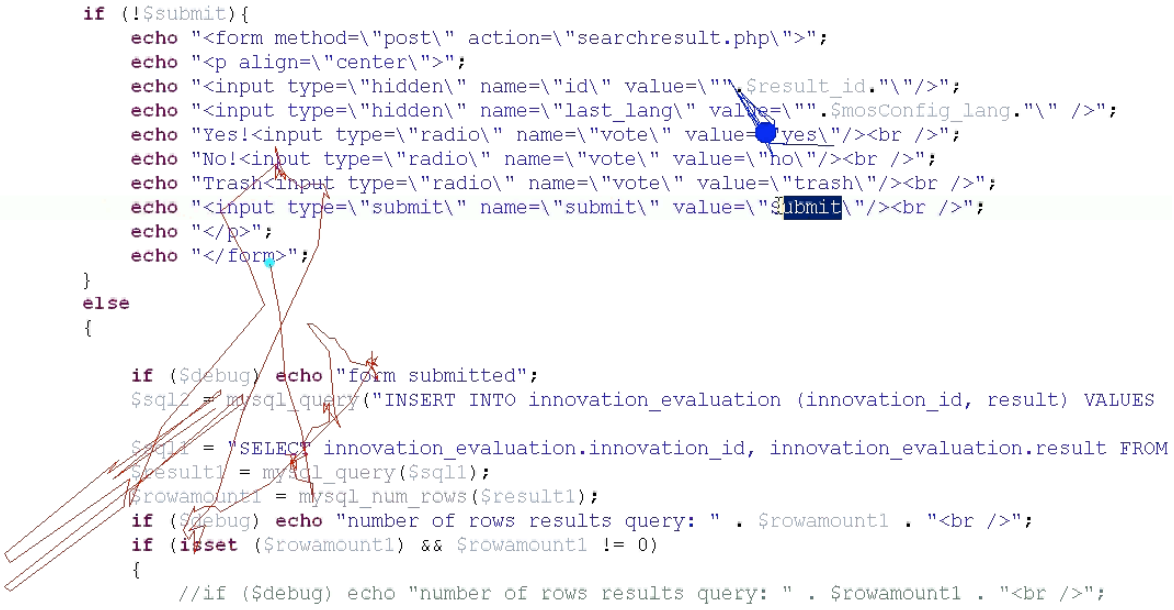

Figure 1. Developer pairs' superimposed visual attention with current point-of-gaze and trailing.

We can see from the previous figure that the pair's visual attention is far from joint, which indicates that their cooperation at that point of time is not very evident. The driver is coding and navigator is reviewing some other part of code. What should be understood here, is that when the collaboration is tighter, i.e. the pair is processing the same part of code - usually the size of a few code blocks - their visual attention seems to be also directed more closer to other developer's gaze. As the close collaboration and communication is important to be able to execute the pair programming session successfully, we suggest that closeness of the pair's gaze, i.e. the level of joint attention, could be possibly used as a metric for evaluation the level (tightness) of collaboration taking account of course that the collaborative work is not always directed with shared or joint attention. This remains still to be validated but offers an opening to the unique set of visual attention metrics of collaborative work.

It might be also possible to resolve, but just in some points of time, which one of the pair worker is executing what role based on just eye movements, because the eye movement patterns and the relative location of gaze in the code editor are different between the roles i.e. contain some role-dependent spatial distribution. It seems that the navigator scans the source lines of code with his or hers gaze with possibly wider spatial coverage, in contrast to driver that concentrates more on some specific lines of code that are under production or change. This might be a result of the reviewer's "scan" pattern in eye movements, which is introduced in Uwano et al. (2006) and characterizes an action in which the reviewer reads the entire code before investigating the details of each source line of code. As the driver usually scrolls the code editor to a more optimal location, these are also possible points to differentiate navigator from driver.

A segment of 41 seconds with gaze trail for navigator and heat map for driver is presented in Figure 2, which exemplifies spatial coverage difference, but the conclusion is better justified using dynamic gaze replay as the current figure exaggerates the difference. There are also eye-tracker model specific characteristics in the eye movements that make the coverage comparison a bit challenging, which clearly supports changing to use just one, or comparable, eye-tracker model for capturing both developer's eye movements. Both pair programmer's eye movement representations are based on

approximately the same length of raw eye-movement data segment, but there is small fixational correction in the driver's eye-movement representation (max fixation length 10 ms) because it could not be removed from the analysis program. Several studies about using eye-movements as a biometric provide promising results (e.g. Bednarik et al. (2005a), which might also function as a partial resolution to this problem.

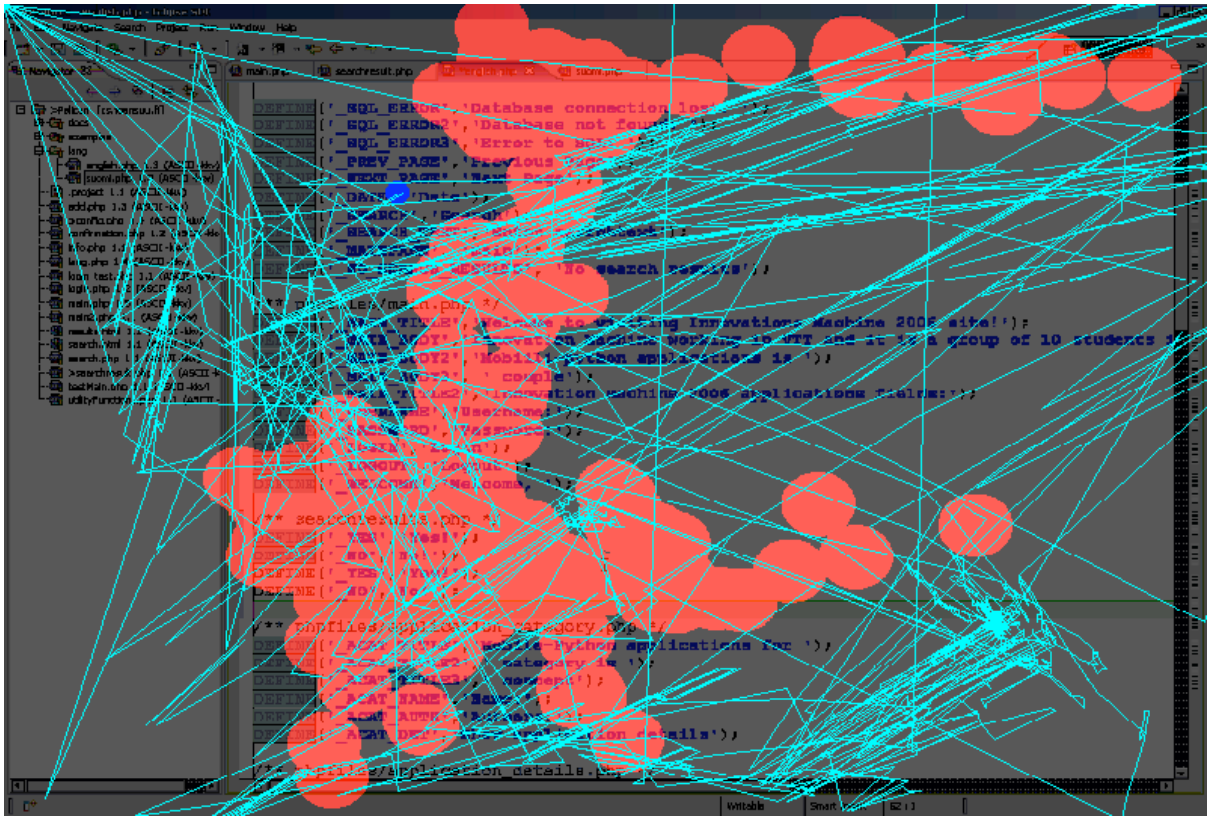

Figure 2. Spatial distribution of visual attention of driver (circles) and navigator (lines). Navigator's visual attention has better coverage of the screen area.

Figure 3 a) represents a longer segment with full gaze path. It tells us that the navigator's gaze trail respect to code editor contains seemly coherent coverage, but the coverage is slightly better in the left side of the screen. In Figure 3 b) heat map highlights this fact well.

a) b)

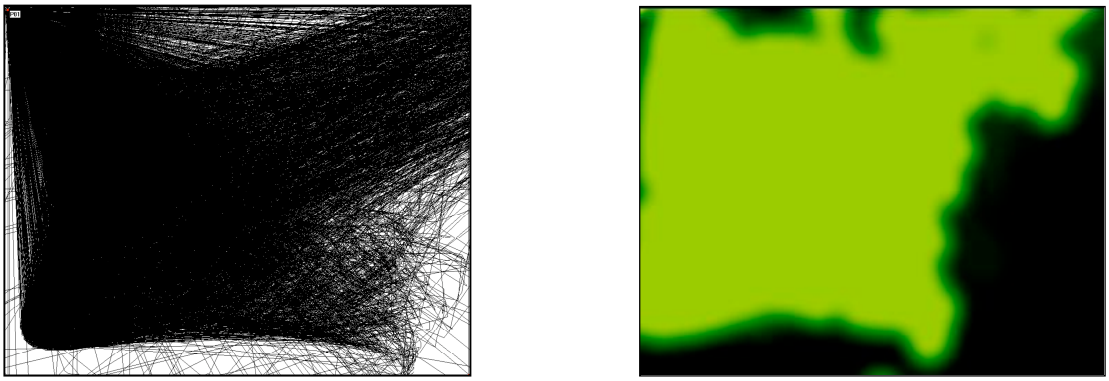

Figure 3. a) Cumulative gaze trail of the navigator in a longer analysis segment (Pietinen, Tenhunen and Tukiainen, 2009) b) Heat map from same segment. Fixations are identified based on pupil size.

To be precise, the code editor contains also a file-explorer, so the previous pictures' weighting on left side should be actually interpreted comparing the whole left side of the screen to the right side of the

screen. This effect is in line with the current view of code reading patterns of programmers and acts also as a partial sanity check for the recorded data. In contrast, the spatial coverage of the driver's visual attention seems to be targeted more into the centre and upper parts of the code editor as he can usually scroll the part of code under processing to that more optimal location for viewing. As eye-tracking data is usually very sparse, the analysis needs to be brought down to very short segments.

Although the focus of this investigation is on programming task, we realize that PP can be and actually was used in other tasks too. Based on the eye-tracking data, the target of the continuous review process, done by the navigator, can be three fold:

1. the program code that is just under writing
2. the program code that is just relevant to the same context (code-block same or near) with the new code and therefore needs to be understood by the pair, and finally,
3. the program code that is visible on the screen, but not relevant for the implementation task at hand.

Elaboration on findings. As there is very little to none of those events, where navigator leads the joint attention to the last case – to the case where the navigator's focus is on those parts of the code that are not directly relevant to the same task that the driver is pursuing on – this particular task done by the navigator does not seem to be beneficial and this suggests that the most efficient review target to be just the new code or the code directly relevant to the task at hand.

The newly produced code should be always reviewed, but how about the parts of the code that are not directly relevant for the current task? With current task there can be multiple source files that gets reviewed as there can be multiple points in program execution that needs to be modified, but with irrelevant parts respect to current task, one can probably review only few parts of some bigger whole, i.e. few parts of a program feature, and there is a good change that those parts are already reviewed earlier.

Partial use of PP can leave un-reviewed source parts to the code base and in this regard it would make sense to review also those irrelevant parts of the source code to the current task, but again, it would still be only few parts of some bigger whole leaving the rest of the parts without review and there is still change that the code gets reviewed twice. This suggests that partial or discontinuous use of PP makes it necessary to mark the parts of the code that should be reviewed in the near future and for example marking by accuracy level of source file would not prevent double reviewing. When pair rotation is added to the picture, the situation gets even more complicated, as there is suddenly bigger lack of knowledge of the parts of the code that are already reviewed. It should be noted here that changing the roles introduces some of the same problems than role rotation, but in lesser extend. The newcomer does not have – or has only partial - memory image of the parts of the code that still needs a review or about the other parts of the code that are not currently visible on the screen but relevant to know to understand the currently reviewed parts of the code, especially e.g. in the case of bad application programming interface and bad division of responsibility.

To sum up, partial use of PP introduces often-ignored challenges to uphold of complete review coverage without double reviewing some parts of the code and there is some extra effort in getting familiar with the system. For knowledge management perspective this might actually be a good thing, but from pure programming productivity perspective bad. One should probably review only the parts the code that are most relevant to on-going task and rotate the pairs only after completion of a task, or even better a feature, for maximal efficiency and systemacy. Another option would be to more actively acquire the driver's attention to those parts of the code that are unfamiliar to navigator, but this would take the focus off from the current implementation task. Another compromise would be to allow some reviewing of the code to be done twice and mark the review needs in file or class level. Retrieving the parts that have been changed from version history might also make the review process more efficient. Clearly, the continuous use of PP is much simpler in what comes to managing the review process.

5. Discussion and Conclusions

To counter the suspects that the advantages or disadvantages of pair programming are not visible in controlled studies and simplified settings with short programming tasks, we decided to use a controlled case study method to study the pair programming practice. Setting up the research environment to acquire the visual attention data from two persons requires more effort than in the typical case of just one person, but it provides insight on how development tasks are carried out in pair programming and what the developers are focusing on during programming.

The Psychology of Software Engineering. Cognitive aspects of programming can be handled under the topic psychology of programming, but when we include the social aspects of programming in PP -- as collaboration is a social phenomenon -- we should refer to sociology and social psychology. Keeping in mind that we are now interested in computer science and its subfield (or parallel field), software engineering (SE), we finally end up with a proper collective topic, the psychology of software engineering (PoSE). This specific research and application area of cognitive psychology is at high importance to the field of software engineering for being able to apply cognitive mechanisms to some higher-order principles - found by SE-research but unfortunately often detached from cognitive psychology - leading to a more predictive and complete understanding of the phenomenon at question.

One example of these is e.g. the learning effect in PP. There is evidence that PP will lead to an increased learning (e.g. McDowell et al., 2003; Mendes, Al-Fakhri and Luxton-Reilly, 2005; Nagappan et al., 2003), but how this actually happens? Cognitive psychology has revealed us that in the process of learning, it does not matter if it is intentional or incidental. So, whether person intends to learn or not, does not matter. The most significant factor is how we process the material (Anderson, 2000). Rehearsal and elaborative processing represents the activities that are conductive to good memory i.e. a good level of processing respect to high learning. In PP, there is lots of elaboration in communication between the developer pair for being able to keep up the shared understanding of, among others, the code or design.

One of the goals of Psychology of Programming (PoP) research is to improve efficiency, which is one determinant of productivity. Productivity is largely a result of people factors like experience and individual human variation (Boehm, 1981), so people factors are the things we need to put more weight on when doing software engineering research.

Continuous Review and Partial Use of PP. Based on our experience, the full use of PP might be sometimes difficult to achieve (Pietinen, Tenhunen and Tukiainen, 2008) and therefore the review task is incomplete. The current description of the protocol leaves much to chance and is therefore inadequate. Without existing better description of the protocol, it should be at least stated that the spare times when the navigator‘s engine is on a free run, he or she should effectively identify these situation and make corrective initiatives to do something actually useful, like doing some up-front design or test cases. Attention should be given also to what parts the code should be reviewed. We suggest that only the code that is relevant to the on-going task should be reviewed during collaboration in shared workspace. The navigator might be also able to fill the spare time by reviewing some other parts of the code, but this might be only efficient with separate workspace in close proximity for better controllability of the workspace, in addition to doing e.g. design and test cases. Will this make obsolete the additional (peer-) review or inspection that is otherwise needed, would be interesting to see. In general, only full time PP use ensures a total coverage of the review process without additional (peer-) review sessions, or other complementary methods, and therefore in partial use, one should keep track on which files are reviewed and which not. This can be done e.g. by including the information to the review target’s version history.

Replacement Options for PP. Another interesting option to PP is to use Side-by-Side Programming (SbSP) introduced in (Cockburn, 2005). It differs from the PP by computer and task allocation: two developer team works in close proximity but with own computers and they have a shared goal, which is just divided into sub-tasks capable of doing individually. In Nawrocki et al. (2005), SbSP resulted in 20% effort overhead compared to two individual workers, but according to authors, close proximity and shared goal might lead into enhanced communication and through that, to better knowledge management. Also, some practitioners seem to have also applied distributed PP in a co-located setting

for overcoming e.g. the ergonomic issues with using just one computer for doing PP. As stated earlier, reviews might also be an option to PP.

Use of Eye-tracking to Study PP Protocol. We propose the use of eye-tracking as a method to study the protocol followed by the developer pair, as it offers a way to overcome or at least diminish some restrictions inherent in think-aloud methods when applied in a task that have an collaborative dimension and also in general. In collaborative work, the think-aloud method inhibits natural turn taking in verbalisation of thoughts, but the eye tracking can be done concurrently for both developers. In general level, think-aloud method can require frequent interruptions from study controller to instruct participants to verbalize more actively, but there is no such requirement with eye-tracking. Then again, eye-tracking contains some limitations that are not problems to same extend with think-aloud method, like being able interpret why the participant is doing something and not just what he or she is doing. Sometimes the question what the participant is doing is reduced just to where he or she is looking, because we do not have an answer to the question why first.

Eye-tracking research in collaborative settings can provide many new insights to the way we are working as pairs or as a team and provide feedback on how well we actually did. It provides a more objective way for evaluating human activity, because in self-reports the activity might be reported based on the perception on how the work should be done and not on how it has actually been conducted. Eye-tracking, especially in conjunction with complementary protocols, can bring new source of evidence when studying human behaviour, thinking and activities, which can be then be used to improve our ways of working and through that, to improve the productivity of the work. This approach can also bring the analysis to the level of detail, where we are not just measuring the productivity (e.g. as an average completion time of different groups), but rather we can take into account the behavioural richness of humans on individual level. On the other hand, at the moment we use eye-tracking as an additional source of evidence when studying PP, as it cannot replace but rather complements the other data sources.

5.1 Future Work

As eye tracking in collaborative settings is fairly new and largely unexplored research area, there is lots of work to be done on improving the eye-tracking technology, analysis methods and tools. New measures and metrics need to be developed, and their relationship to the way programmers work need to be evaluated. As human-related issues seem to be the topics that are not given enough attention in SE research, eye tracking in collaborative settings can bring us new knowledge also in this regard. Support for automatic synchronization of multiple recordings is needed, but it is missing at the moment from the environment described in Pietinen, Tenhunen and Tukiainen (2008). The replacement of the manual method is needed for better accuracy in timing.

Regarding near future, we plan to continue in the analysis of visual attention in PP and accommodate additional data sources. For future application, Salinger and Prechelt (2008) offers a set of concepts resolved based on grounded theory, which can be used for describing what happens during PP. In addition, we plan to investigate the cognitive load during PP using eye tracking. Also the capability of defining Areas of Interests (AOI's) in 3D environment in eye tracking setup introduces interesting research avenues making it possible to efficiently conduct statistical analysis on the targets of attention in the whole working environment, such as the other members of the development team.

The joint attention could possibly be triggered more efficiently without finger pointing, or other body gestures, by super-imposing both developers eye movements to the monitor screen. In Brennan et al. (2008), support is provided for improved performance using shared gaze during collaborative search. This could enhance the collaboration especially in distributed settings where there is limited possibility to use nonverbal communication methods (FaceTop discussed e.g. in Stotts, Smith and Gyllstrom (2004) is an effort to fix this situation) and when used locally; improve the ergonomic problems related to working with single monitor. Without a doubt, these ideas also introduce some new questions to be answered.

References

Anderson, J. R. (2000) Cognitive psychology and its implications. 5th Ed, Worth Publishers, NY.

Arisholm, E., Gallis, H., Dybå, T., Sjraberg, D. (2007) Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. IEEE Transactions on Software Engineering 33(2), pp. 65–86.

Baheti, P., Gehringer, E., Stotts, D. (2002) Exploring the Efficacy of Distributed Pair Programming. In Proceedings, XP/Agile Universe 2002, New York, Springer, Heidelberg, pp. 208–220.

Bednarik, R., Tukiainen, M. (2008) Temporal Eye-Tracking Data: Evolution of Debugging Strategies with Multiple Representations. In Proceedings of the 2008 Symposium on Eye Tracking Research & Applications (Savannah, Georgia, March 26 - 28, 2008). ETRA '08. ACM, New York, NY, pp. 99–102.

Bednarik, R., Tukiainen, M. (2007) Analysing and Interpreting Quantitative Eye-Tracking Data in Studies of Programming: Phases of Debugging with Multiple Representations. In Proceedings of the 19th Annual Workshop of the Psychology of Programming Interest Group (PPIG'07), Joensuu, Finland, July 2-6, 2007, pp. 158–172.

Bednarik, R., Kinnunen, T., Mihaila, A., Fränti, P. (2005a) Eye-Movements as a Biometric. In Proceedings of the 14th Scandinavian Conference on Image Analysis, SCIA 2005, Joensuu, Finland, June 19-22, 2005. Lecture Notes in Computer Science, Vol. 3540, Springer, 2005, pp. 780-789.

Bednarik, R., Myller, N., Sutinen, E., Tukiainen, M. (2005b) Effects of Experience on Gaze Behaviour during Program Animation, In proceedings of the 17th Annual Psychology of Programming Interest Group Workshop (PPIG'05), Brighton, UK, June 28 - July 1, 2005, pp. 49–61.

Brennan, S. E., Chen, X., Dickinson, C. A., Neider, M. B., Zelinsky, G. J. (2008) Coordinating cognition: The costs and benefits of shared gaze during collaborative search, Cognition, Volume 106, Issue 3, March 2008, pp. 1465-1477.

Boehm, B.W. (1981) Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, pp. 427–444.

Bryant, S., Romero, P., du Boulay, B. (2008) Pair programming and the mysterious role of the navigator, Int. J. Human-Computer Studies, Vol. 66, No. 7, July 2008, pp. 519–529.

Chong, J., Hurlbutt, T. (2007) The Social Dynamics of Pair Programming, Proc. 29th Int. Conf. on Software Engineering, IEEE Computer Society, pp. 354–363.

Ciolkowski, M., Schlemmer, M. (2002) Experiences with a Case Study on Pair Programming, In Workshop on Empirical Studies in Software Engineering.

Cockburn, A. (2005) Crystal Clear: A Human-Powered Methodology for Small Teams, Addison-Wesley.

Constantine, L.L. (1995) Constantine on Peopleware, Yourdon Press Computing Series, ed. E. Yourdon. Englewood Cliffs, NJ: Yourdon Press.

Crosby, M.E., Scholtz, J., Wiedenbeck, S. (2002) The roles beacons play in comprehension for novice and expert programmers, In Proceedings of the 14th Annual Psychology of Programming Interest Group Workshop (PPIG'02).

Crosby, M.E., Stelovsky, J. (1990) How do we read algorithms? A case study, IEEE Computer, vol. 23, no. 1, pp. 24–35.

Damiani, E., Gianini, G. (2007) A Non-invasive Method for the Conformance Assessment of Pair Programming Practices Based on Hierarchical Hidden Markov Models, Lecture Notes in Computer Science. Springer Berlin / Heidelber, 4536/2007, pp. 123–136.

DeClue, T. (2003) Pair programming and pair trading: effects on learning and motivation in a CS2 course, The Journal of Computing in Small Colleges, 18 (5), pp. 49–56.

Duchowski, A.T. (2003) Eye Tracking Methodology: Theory & Practice, Springer-Verlag, Inc., London, UK.

Ericsson, K., Simon, H. A. (1993) Protocol Analysis: Verbal Reports as Data, Cambridge, Mass, MIT Press.

Flor, N. V. (2006) Globally distributed software development and pair programming, Comm. ACM 49, 10 (2006), pp. 57–58.

Hulkko, H., Abrahamsson, P (2005) A multiple case study on the impact of pair programming on product quality, In Proceedings of the 27th international conference on Software engineering (ICSE), pp. 495–504.

Höfer, A. (2008) Video analysis of pair programming, In Proceedings of the 2008 international Workshop on Scrutinizing Agile Practices Or Shoot-Out At the Agile Corral, Leipzig, Germany, May 10 - 10, 2008, APOS '08, ACM, New York, NY, pp. 37–41.

IEEE Std. 1028-2008 (2008), Standard for Software Reviews and Audits, The Institute of Electrical and Electronics Engineers, Inc.

Just, M.A., Carpenter, P.A. (1980) A theory of reading: From eye fixations to comprehension, Psychological Review, 87(4), pp. 329–254.

Lui, K., Chan, K. (2006) Pair programming productivity: Novice-novice vs. expert-expert. International Journal of Human-Computer Studies 64(9), pp. 915–925.

McDowell, C., Werner, L., Bullock, H.E., Fernald, J. (2003) The impact of pair programming on student performance, perception and persistence, In Proceedings of the 25th international Conference on Software Engineering, Portland, Oregon, May 03 - 10, 2003, International Conference on Software Engineering. IEEE Computer Society, Washington, DC, pp. 602–607.

McDowell, C., Hanks, B., Werner, L. (2003) Experimenting with pair programming in the classroom, Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education, ACM SIGCSE Bulletin, 35(3), pp. 60–64.

Mendes, E., Al-Fakhri, L. B., Luxton-Reilly, A. (2005) Investigating pair-programming in a 2nd-year software development and design computer science course, In Proceedings of the 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education (Caparica, Portugal, June 27 - 29, 2005), ITiCSE '05, ACM, New York, NY, pp. 296–300.

Müller, M.M. (2005) Two controlled experiments concerning the comparison of pair programming to peer review, Journal of Systems and Software, Volume 78, Issue 2, November 2005, pp. 166–179.

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., Balik, S. (2003) Improving the CS1 experience with pair programming, In Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education (Reno, Navada, USA, February 19 - 23, 2003), SIGCSE '03, ACM, New York, NY, pp. 359–362.

Nawrocki, J.R., Jasiński1, M., Olek, Ł., Lange, B. (2005) Pair Programming vs. Side-by-Side Programming, Lecture Notes in Computer Science, 3792/2005, Springer Berlin / Heidelberg.

Parnas, D. L, Lawford, M. (2003) Guest Editors' Introduction: Inspection's Role in Software Quality Assurance, IEEE Software, v.20 n.4, pp. 16–20.

Pietinen, S., Tenhunen, V., Tukiainen, M. (2009) Productivity and Eye Movements in Collaborative work: Case Pair Programming, Proceedings of the 9th International SPICE Conference on Process Assessment and Improvement (SPICE 2009), Turku, Finland, June 2-4, 2009.

Pietinen, S., Bednarik, R., Glotova, T., Tenhunen, V., Tukiainen, M. (2008) A Method to Study Visual Attention Aspects of Collaboration: Eye-Tracking Pair Programmers Simultaneously, In Proceedings of the 2008 Symposium on Eye Tracking Research & Applications, Savannah, Georgia, March 26 - 28, ETRA '08, ACM, New York, NY, pp. 39–42.

Pietinen, S., Tenhunen, V., Tukiainen, M. (2008) Productivity of Pair Programming in a Distributed Environment – Results from Two Controlled Case Studies, In Proceedings of the European Systems & Software Process Improvement and Innovation (EuroSPI'08).

Preston, D. (2005) Pair programming as a model of collaborative learning: a review of the research, J. Comput. Small Coll. 20, 4 (Apr. 2005), pp. 39–45.

Reifer, D.J. (1997) Practical Software Reuse, 1st edn. John Wiley & Sons, Inc, Chichester.

Romero, P., du Boulay, B., Lutz, R., Cox, R. (2003) The effects of graphical and textual visualisations in multi-representational debugging environments, In HCC '03: Proceedings of the IEEE 2003 Symposia on Human Centric Computing Languages and Environments, Washington, DC, USA: IEEE CS, 2003.

Rostaher, M., Hericko, M. (2002) Tracking Test-First Pair Programming - An Experiment, In Proceedings, XP/Agile Universe, Springer, New York, pp. 174–184.

Sajaniemi, J. (2008) Guest Editors' Introduction: Psychology of Programming: Looking into Programmers' Heads, Special Issue on Psychology of Programming, Human Technology, Volume 4 (1), May 2008, pp. 4–8.

Salinger, S., Prechelt, L. (2008) What happens during Pair Programming?, Proceedings of the 20th Annual Workshop of Psychology of Programming Interest Group (PPIG '08), Lancaster, September 10-12.

Shaochun, X., Rajlich, V., Marcus, A. (2005) An empirical study of programmer learning during incremental software development, Cognitive Informatics, 2005, (ICCI 2005), Fourth IEEE Conference on 8-10 Aug. 2005, pp. 340–349.

Shaochun X., Rajlich, V. (2005) Dialog-based protocol: an empirical research method for cognitive activities in software engineering, Empirical Software Engineering, 2005, International Symposium on 17-18 Nov, pp. 10.

Stotts, D.P., McC. Smith, J., Gyllstrom, K. (2004) Support for Distributed Pair Programming in the Transparent Video Facetop, XP/Agile Universe, Calgary, Canada, August 15-18, pp. 92–104.

Succi, G., Marchesi, M. (2001) Extreme Programming Examined. Pearson Education, London.

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 &Amp; Applications (San Diego, California, March 27 - 29, 2006). ETRA '06. ACM, New York, NY, pp. 133–140.

Vanhanen, J., Lassenius, C. (2005) Effects of Pair Programming at the Development Team Level: An Experiment, In Proceedings of International Symposium of Empirical Software Engineering (ISESE 2005).

Williams, L., Kessler, R. (2003) Pair Programming Illuminated. Pearson Education, London.

Williams, L.A. (2000) The collaborative software process. PhD Dissertation. Department of Computer Science. Salt Lake City. University of Utah.

Williams, L., Kessler, R. (2000) All i really need to know about pair programming i learned in kindergarten. Communications of the ACM 43, pp. 108–114.

Williams, L., Kessler, R. (2000b) "The Effects of "Pair-Pressure" and "Pair-Learning" on Software Engineering Education, 13th Conference on Software Engineering Education and Training, 6-8 March 2000, USA, pp. 59–65.

Williams, L., Kessler, R., Cunningham, W., Jeffries, R. (2000) Strengthening the case for pair programming, IEEE Software 17, pp. 19–25.

Williams, L., Upchurch, R. (2001) In support of student pair programming, Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education, ACM SIGCSE Bulletin, 33(1), pp. 327–331.