Gaze Evidence for Different Activities in Program Understanding

Kshitij Sharma, Patrick Jermann, Marc-Antoine Nüssli, Pierre Dillenbourg

CRAFT, École Polytechnique Fédérale de Lausanne
<firstname>.<lastname>@epfl.ch

Abstract We present an empirical study that illustrates the potential of dual eye-tracking to detect successful understanding and social processes during pair-programming. The gaze of forty pairs of programmers was recorded during a program understanding task. An analysis of the gaze transitions between structural elements of the code, declarations of identifiers and expressions shows that pairs with better understanding do less systematic execution of the code and more "tracing" of the data flow by alternating between identifiers and expressions. Interaction consists of moments where partners' attention converges on the same same part of the code and moments where it diverges. Moments of convergence are accompanied by more systematic execution of the code and less transitions among identifiers and expressions.

1 Introduction

In the last decade, off the shelf screen based remote eye-trackers have become readily available. These devices offer system designers and social scientists unprecedented access to the users' attention. Our long term goal is to use automatically collected gaze traces to provide feedback to the users and adapt the system to their level of expertise. A prerequisite for such undertakings is that we better understand the relationships between gaze-based behavioural indicators and task-based performance.

In this contribution we propose an analysis of pair-programming that illustrates the sensitivity of gaze traces to different levels of understanding as well as to different modes of interaction. This problematic is a two sided coin: it involves cognitive aspects related to program understanding and social aspects related to the interaction of two programmers.

Program understanding is central in many programming tasks, for example during software maintenance or software evolution where programmers have to read and extend code that they did not necessarily produce themselves. Program comprehension is a goal-oriented, problem-solving task that is driven by preexisting notions about the functionality of the given code [10]. It can be thought of a pattern matching at different levels of abstraction [20]. The different abstraction levels help understanding a program at different levels, for example, at syntactical level programmers can understand the relation between different programming constructs and at semantic level they can relate different programming structures to their real world counterparts. The potential of eye-tracking in diagnosing the quality or the strategies of understanding relies on the assuption that understanding strategies are reflected by different ways to "read" the code.

From the point of view of the interaction between pair programmers, we are interested in finding the effect whether different types of collaboration strategies are reflected by gaze indicators and whether they are linked to a program comprehension strategy. Previous experiments carried out using cross-recurrence [17] as a measure of gaze coupling showed that the higher the coupling, the better the outcomes of collaboration. This might not be true for more specific problem solving tasks. For example, in collaborative program understanding the task requires the participants in collaborative environment to develop their own understanding as well as to agree upon a solution. During an extended pair programming session, programmers do not necessarily attend to the same information at all times. In terms of interaction, this translates to the requirement for different phases in the interaction. During convergent phases, they look at the same part of program. During divergent phases, participants look at different parts of the program. Both of them can contribute to understanding a program in different ways. These phases might reflect different understanding strategies in program comprehension, for example individual hypothesis building and collaborative verification.

In next section we present related work about program understanding and dual eye-tracking for pair programming. We then describe the problem statement and research questions. The Methods section

gives the details of the experiment and the algorithm that was used to find different phases of interaction. We then present and discuss the results from the experiment.

2 Related Work

2.1 Program Understanding

Program understanding is a special kind of problem solving. Like any problem solving task, program comprehension has a problem statement (to understand the given program) and a solution (description of functionality of the program) and different approaches to get the solution. The main approaches include top-down and bottom-up. Top-down approach involves decomposition of the problem in sub-problems and solving the sub-problems, while bottom-up approach involves integration of low-level details to come up with a solution.

J.Larkin [11] found in her studies about solving physics problems two approaches of problem solving; first, to approach the solutions from the problem givens(top-down) and second, to backtrack from the solution to the problem givens(bottom-up). Similarly, J.R. Anderson [1] found in his studies two ways to write programs; first, writing code line by line(bottom-up) and second decomposing the goal of the program into subgoals and implementing the subgoals(top-down).

In the case of program understanding, there are many strategies to understand a program, a top down approach [19] consists of starting with a hypothesis about the program and then validating or "end marking" the hypothesis with the individual components of the program. A Bottom-up approach [18] starts from a series of code fragmentation and then assigns a domain concept to each fragment. An Iterative approach [4] includes a "while" loop of top-down process, i.e., having a set of preexisting notions or hypothesis, their verification and modification, until everything in the program can be explained within the set of notions with which the iteration started. There are some more strategies that are a hybridisation of top-down and bottom-up [13] [14]. These two strategies are used interchangeably during program comprehension as and when needed [13].

S. Letovsky [13] proposed a typical set of mental models needed to understand a program which includes specification, implementation and annotation of different parts of the program. [13] also emphasised that mental model for implementation consists of actions and data structures of a program. Understanding the entities/data/variables and relationship amongst them inside a program is very important in order to assign them a concept from the domain knowledge [21]. [8] advocates for having a programming plan to understand the program text (what is written?) and the program intent (why is something written?), and then divides the programming plan into two major parts "Variable plan" and "Control plan". [8] then proposes the use of variable plan to understand the relation between program text and program intent.

In two different studies [3] and [10] describe the particular strategies for novice and expert programmers respectively. On one hand, [3] finds that for the understanding of novices while loops sometimes become "while demons". Novices have "conflicts" in the strategies to be applied for giving the "Natural Language Description" of a program. Novices tend to follow the "systematic execution" of the program and increase their chances to get stuck. On the other hand, [10] finds that experts go for "as-needed" strategy, where they limit their understanding to only those parts of the program that they find relevant to a given task. Experts do not follow a predefined strategy to understand a program. For example, experts do not decide beforehand to understand a program in "top-down" or "bottom-up" manner. Experts tend to use both of them as and when needed.

2.2 Dual Eye Tracking for Pair Programming

Two synchronous eye-trackers can be used for studying the gaze of two persons interacting to solve a problem. It gives a chance to understand the underlying cognition and social dynamics when people collaborate. Nüssli [15] gives a two way motivation for dual eye-tracking. Using statistics to find the relation between the gaze features and collaboration events and using machine learning for prediction of some collaboration attributes from gaze patterns. In their study of collaboration amongst a pair Richardson and Dale [5] found that when two persons talk about something they see, they tend to look at the

same thing in the stimulus. [17] measured the "togetherness" of the participants in a "speaker-listener" pair using cross-recurrence plots and found that when the listener follows the gaze of the speaker (s)he had a better comprehension. This idea of "looking together" may not be true to a pair of programmers looking at a program and trying to understand it because the task of program understanding is more specific than the task of listening to a speaker. "Togetherness" of their eye-movements can help to define different phases of interaction the pair undergoes during the task of program comprehension. These different phases occur when the participants are looking at the same part of the program as opposed to the case when they are not. Both types of the phases play there role in the understanding of a program as we mentioned in the introduction.

Pair programming is usually done with co-located programmers. They play the roles of driver (actual typing) and navigator (more like a organisational activities). Spatially distributed pair programming have been studied with satisfactory results showing that the distance factor can be neglected [2]. Pair programming leads to high quality programs [15], hence a pair of expert programmers can obtain a better understanding of a program as well.

3 Problem Statement

3.1 Program Understanding and Gaze Transitions

Is it possible to detect different strategies for program understanding between the pairs with perfect versus low levels of understanding ? Do they build their understanding based on different semantic elements in the program than the pairs with the low level of understanding? There are many ways to go about solving the problem of program understanding as we mentioned in the related work. We also mentioned that program understanding strategies are different for the people who have better understanding than others who don't. We are interested in finding this difference in terms of their eye-movements.

To measure exploration strategies, we adopt an approach based on gaze transitions between different types of program elements. More precisely, as a stimulus for eye-tracking, a program can be divided in three main semantic classes (or Areas of Interest). These semantic classes are identifiers (I), structural (S) elements and expressions (E) in the program. Typically, identifiers are the variable declarations, structural elements are the control conditions of the program and expressions represent the data flow and relationship amongst the variables. Thus we propose to say that a "to and fro" shift in gaze between identifiers and expressions will depict the attempt to understand the data flow and/or the relation among the variables. Similarly, a gaze shift among all the three semantic classes will translate in an effort to understand the data flow according to the conditions in the program. In terms of program understanding strategies this behaviour is attributed to "Systematic program execution".

Our analysis aims at finding which type of transitions characterise pairs with different levels of understanding. Table 1 shows the categorisation of different transitions among different semantic classes in the program into data flow, control flow and data flow according to control flow. We consider "3-way" transitions among the semantic classes as one 3-way transition reflects one unit of program understanding behaviour. For example, a 3-way transition "E$-$ $>$I$-$ $>$E" reflects the "reference lookup" for a variable in an expression.

Question 1 Is there a relation between the transitions among different semantic classes in the program and the levels of program understanding ?

3.2 Convergent and Divergent Episodes of Interaction

Collaboration consists of a series of convergent and divergent phases. When partners work as a team and put their joint efforts to understand the code we say that they are convergent in their interaction and we have a convergent episode of interaction. In a convergent episode participants in a pair look at the same part of the program in a "stable" manner. "Stable" manner of looking at a program is reflected by fixations in a small range (less than a threshold) of tokens (see section "segmentation of eye-tracking data" for more details). On the other hand when the participants try to build their own understanding and they are looking at the different parts of the program, we say that they are diverging and we have a divergent episode of interaction.

Table 1: Categorization of different transitions among different semantic classes in the program into different types of flows in the program. (I=Identifier, S=Structural, E=Expression). $-$ $>$ denotes transition.

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

The basic question related to episodes is whether individuals use different reading strategies during convergent and divergent episodes. A complementary question is whether pairs with different levels of understanding behave differently in convergent and divergent episodes. To define convergent and divergent episodes of interaction, we need to segment the eye-tracking data for individual participants and then align them in time; so that we can compare the segments of the two participants on the scale of vicinity in terms of fixations.

Question 2 How do convergent and divergent phases of interaction affect the program comprehension strategies of pairs with different levels of understanding?

3.3 Segmentation of interaction

The segmentation of interaction into segments or ”meta-fixations” during which attention is focused on a stable set of objects is a novel approach in gaze analysis. Usually, fixation time is aggregated in predefined areas of interest and researchers report global proportions of attention time dedicated to the different types areas. To measure coupling, cross-recurrence analysis quantifies as a global measure how much the gaze of the collaborators follow each other with a given lag. These fixation based measures aggregate indicators measured in the 100ms range to the whole duration of the interaction. The segments that we propose to detect on the other hand are situated in between the short time range of a fixation and the long time span of the whole interaction. Figure 1 shows the conceptual difference between the fixations and segments. The main difference is in their respective durations in time and their use to analyse different types of behaviours.

We considered different time series segmentation algorithms [16] [9] [22] to implement a method that would segment the date into meta-fixation but none of these methods carried the notion of meta-fixations or a hierarchy of segments in terms of their duration. Hence, we computed the segments from the fixations using a very simple procedure. This procedure computes segments from the fixations in a similar way as fixations are computed from the raw gaze data. Moreover, none of the methods for segmentation or change point detection describe method for finding the segmentation on two simultaneous time series and to align the segments in terms of time. We give details of this procedure in next section.

4 Methods and Materials

4.1 Experiment

In the experiment, pairs of subjects had to solve two types of pair programming tasks. The first task was to describe the rules of a game (e.g., initial situation, valid moves, winning conditions, and other rules) implemented as a Java program. The only hint to the pairs was that it is a turn based arithmetic game. The second task was to find errors in the game implementation and to suggest a possible fix using a few lines of output to analyse the error and to find the location of it in the code.

Subjects Eighty-two students from the departments of computer science and communication science of the École Polytechnique Fédérale de Lausanne, Switzerland were recruited to participate in the study.

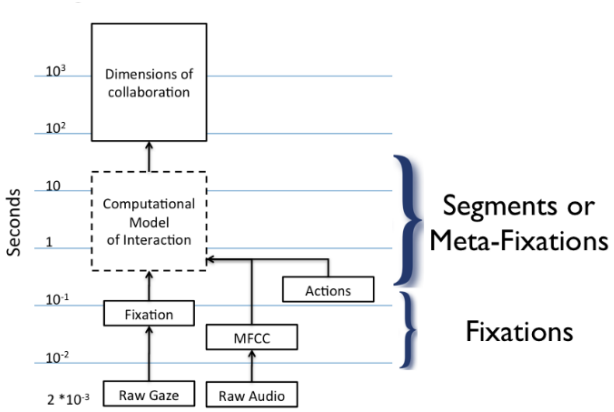

Figure 1: A typical Diagram to show the conceptual analogy between the fixations and the segments, and to show the analogy between different levels of raw gaze aggregation and the behaviour dimensions

They were each paid an equivalent of 20 USD for their participation in the study. The participants were typical bachelor and master students. The participants were paired into forty pairs irrespective of their level of expertise, gender, age or familiarity.

Procedure Subjects had to read and sign a participation agreement form, when they came to laboratory. Then, for the next 3 minutes, the experimenters calibrated the eye-trackers for each of the subjects. This simple procedure consists of fixating the centre of nine circles appearing on the screen. Once both subjects were ready, they individually filled a short electronic questionnaire about their programming skills and previous experience. The pretest which followed, consisted of individually answering thirteen short programming multiple choice questions.

Apparatus and Material Gaze was recorded with two synchronised Tobii 1750 eye-trackers that record the position of gaze at 50Hz in screen coordinates. The eye-trackers were placed back to back and separated from each other by a wooden screen. The synchronisation of the eye-trackers was done by using a dedicated server to log gaze via callback functions from the low-level API of the eye-trackers. The subjects heads were held still with an opthalmologic chin-rest placed at 65 centimetres of the screen. An adaptive algorithm was used to identify fixations and a post-calibration was done to correct for systematic offsets of the fixations with regards to the stimulus (see [10] for details about these procedures).

The JAVA programs were presented in a custom programming editor based on the Eclipse development environment. Text was slightly larger (18pt) than it is usually on computer screens and was spaced at 1.5 lines to facilitate the fixation hit detection at a word level precision. Scrolling was synchronised between the participants, such that when programmers scrolled, their partners' viewport was also updated at the same time. All other highlighting, search and navigation functionalities were disabled in the editor.

Level of Understanding We distinguish between three levels of understanding based on how well they performed the description task.

Good Pairs with a good understanding are able to describe the the rules of the game (initial situation, valid moves, winning conditions).

Medium Pairs with a medium level of understanding only describe partial aspects of the game structure, and often give algorithmic descriptions of the program and try to guess the detailed rules from the method names; but they failed to get the winning condition.

Poor Pairs with a poor understanding are not able to describe the functionality of the code. During analysis we saw that the pairs with poor level of understanding were only those pairs which had novices as both the participant and their gaze pattern was as good as a reading plain text and not a program.

Moreover, they could explain only some of the the syntactical things properly (e.g., they say that there is a while loop that runs until some condition but could not explain what it is doing), so we decided to analyse with only two levels of understanding.

Tokens, Semantic Classes and Transitions The program is comprised of tokens. For example, a line of code "int i = 5;" contains 8 tokens (int, i, =, 5, ;, and 3 spaces). We recorded the time spent on the various tokens in the program and categorised them into three semantic classes:

Identifier this class includes the variable declarations.

Structural this class includes the control statements.

Expression this class includes the main part of the program, like the assignments, equations, etc.

We took the change of gaze from one semantic class to a different class as the transition between the semantic classes. Here, we do not consider the change of gaze position from one token of a particular semantic class to another token of the same class.

4.2 Segmentation of Eye-Tracking Data

In this section we present the approach to aggregate the fixations into the segments. The existence of segments first came to our attention when looking at the evolution in time of the JAVA tokens looked at by the programmers during a program understanding task. The curve in the figure 2 represents the evolution of the average token identifier in time (tokens were numbered in order of appearance in the code), for a particular pair. Stable exploration episodes clearly appear as "plateaux" separated by "valleys" and are reminiscent of the data patterns that characterise the organisation of raw gaze data into fixations and saccades. Deep valleys are due to programmers scrolling through the code while looking for particular methods whereas smaller valleys correspond to focus shifts between areas of code visible on one screen. Segmenting the fixation data into interaction episodes is a two step process; first we find the segments for individual participant in the pair and then we align them in time to find the episodes of interaction for the pair.

Finding Segments in the Gaze of Individual Participants For finding the segments from the individual fixation data, first of all we smooth the fixations using moving averages; and then used the following steps to find the segments from the individual fixation data:

1. Divide the smoothened fixation data into non-overlapping windows.
2. For fixations in each window find a best fitting line.
3. For each fitted line find the angle it makes with the time axis and for each window find the range of tokens looked at by the participant.
4. For each window find whether the angle between the line and the time axis and the range of tokens looked at are both less than the respective thresholds; if yes, then the window is deemed to be a part of a segment.
5. once we have the potential portions of a segment; we merge such windows that are consecutive in time, only if they are overlapping in terms of the range of tokens looked at.
6. The output of this step is the segments or the merged windows for both of the participants in a pair.

Figure 2 shows the segments computed from the fixation data (sampling rate 5Hz) for two participants in the same pair. The black lines depict the segments. These individual segments are used to define a set of different episodes of interaction during the whole interaction, we describe this step in next section.

Temporally Aligning the Segments for the Pair Using the segments for both the participants we align them in time and then again merge the segments so that we have longer (in terms of time) episodes of interaction to analyse.

For finding the episodes of interaction we use the following steps:

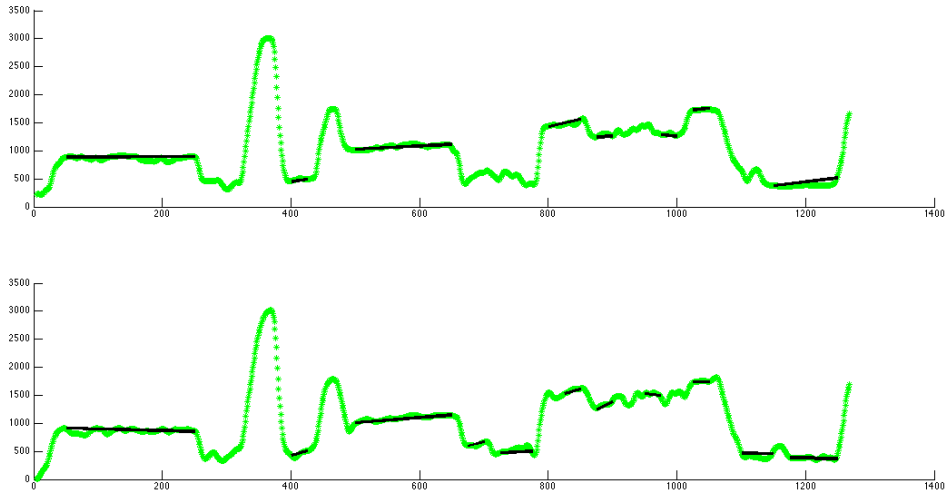

Figure 2: Segments computed for individual participants of a pair in the program understanding task. The x axis represents time (sampling rate 5Hz). The y axis represents the average token ID that was gazed at. A horizontal "plateau" (black horizontal lines) means that the subject has been looking at a stable range of tokens.

1. Input to this step is the segments for both the individuals in a pair that we get as the output of the previous step.
2. For each segment of one participant find the temporal overlap of this segment with each of the segments of the second participant and make a binary overlap matrix where each element indicating whether the i^{th} segment of first participant overlaps (more than a threshold) with the j^{th} segment of the second participant,in terms of time and the range of tokens looked at (intuitively we can say that there is no temporal overlap between the non-consecutive segments).
3. Once we have the overlap matrix, we take the intersection of the segments for the two participants (in terms of their duration) and define the intersection to be the convergent episodes of interaction.
4. The output of this step is the set of convergent episodes of interaction for a pair.

Figure 3 shows an example of temporal alignment of the segments for the two participants in a pair and the episodes of interaction in terms of time. The episodes of interaction are then used to form a Contingency Table (see section 5.1) which is used to analyse the gaze inside an episode and overall interaction. We give details about the contingency tables and the analysis in next section.

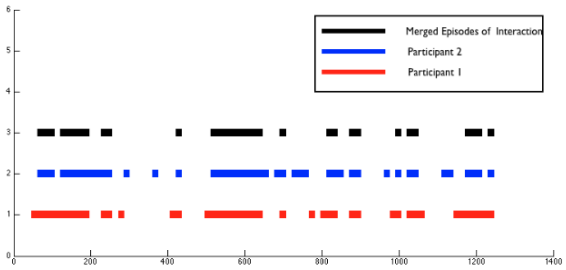

Figure 3: Segments of both the participants aligned in time and the episodes of interaction; time on X-axis; Y-axis: 1 for first participant, 2 for the second participant, 3 for the episodes of interaction

4.3 Data Preparation for Analysis

In this section we describe the pathway from raw gaze data to the contingency tables of transition between the semantic classes.

Raw Gaze and Fixations Raw data from eye-trackers come at a high sampling rate that is well above (typically at 50Hz and higher rates) the rate of gaze fixations. Hence, the first step in the analysis of gaze aggregates the gaze points given by the eye tracker into fixations (moments of relatively stable gaze positions).

Determining Areas of Interest : Tokens Once we have the fixations from the raw gaze data we define the areas of interest in our stimulus i.e., in the program.

Episodes of Interaction From the fixations we get the episodes of interaction using method described in section "segmentation of eye-tracking data".

Tokens to Semantic Classes After defining the tokens as our areas of interest we categorised them in 3 categories Identifiers, Structural and Expressions (see section "program understanding and gaze transitions" for details).

Sequence of Semantic Classes looked at We took the sequence (time series) of the Semantic classes fixated during the interaction for our analysis (we took the data inside and outside of the "segments" while analysing convergent and divergent episodes respectively), for example sequence "IIIESSEESSSIIIE" (I = Identifiers, S = Structural and E = Expressions) tells us that first 3 fixations were on identifiers, 4^{th} fixation was on an expression then next 2 fixations were on the structural elements and so on.

Lumping of Sequence As we are interested in the transitions between the semantic classes and not in the duration of time spent on the different semantic classes. We took the continuous fixations on the same semantic class to be one fixation and thus the above example sequence turned into a "lumped" sequence as "IESESIE".

Lumped Sequence to "3 way" Transitions Once we had the lumped sequence we simply counted the number of transitions from one semantic class to other and then to another one. For example the lumped sequence "IESESIE" has 5 transitions "IES", "ESE", "SES", "ESI" and "SIE".

Transitions to Control Flow Transitions "ISI" and "SIS" depict the activity of tracing the control of the program with the different states of the variables.

Transitions to Data Flow Transitions "IEI" and "EIE" depict the activity of tracing the data flow of the program. This reflect the task of looking for different variables and their interdependencies.

Transitions to Systematic Program Execution All the transitions involving the three semantic classes and the transitions "ESE" and "SES" reflect gaze transition amongst all the semantic elements in a program. This translates to the task of considering the modification of an entity as per the control flow of a program.

5 Results

5.1 Question 1

We return to our main interest of finding the difference between gaze transitions for the pairs with different levels of understanding. We first report a relation between the level of understanding of the pair, the pair composition and the gaze transitions using log linear models [6]. Log linear models use contingency tables [12] to find the relation between different variables and for comparing the two models for same contingency table [6] used a new statistics, called G^2 the "likelihood statistics" (or LRX^2), which is asymptotic to "chi square". G^2 can be calculated as following:
$G^2 = 2\sum_i (observed)_i log\frac{(observed)_i}{(expected)_i}$

There are two main methods for fitting the log linear model to a given contingency table. "Forward Selection", where we fit all hierarchical models that include the current model and differ it by one effect; and "Backward Elimination" leaves the term that incurs the least change in the LRX^2 value (for details see [6]). We combined both of the methods to achieve a fast consensus. According to the forward selection we fit all the hierarchical models that differ the current model by one term; and for the next

iteration we keep the model with least change in the LRX^2 value (opposite to the backward elimination, but the idea is to delete the least change incurring term). The finally selected must have the maximum degrees of freedom with the least change in the "likelihood statistics" (or LRX^2).

Table 2: Hierarchical linear model fitting for Contingency Table with dimensions Transition (T), Pair Type (P) and Level of Understanding (UND), for the combined gaze of all the pairs

Model	G^2	DoF	Terms Deleted	$\triangle G^2$	$\triangle DoF$
$[T][P][U]$	7503	57			
$[TPU]$	0	0			
$[TP][TU][PU]$	32	22	$[TPU]$	32	22
$[TP][TU]$	7267	24	$[PU]$	7235	2
$[TP][PU]$	103	33	$[TU]$	71	11
$[TU][PU]$	54	44	$[TP]$	22	22
$[TU]$	8026	48	$[PU]$	7972	4
$[PU]$	8487	66	$[TP]$	8433	22

Table 2 shows the log linear model fitting using the method proposed above. The first 2 models $[T][P][U]$ and $[TPU]$ are the "independence model" and the "saturated model" respectively. We can see that the saturated model fits the data perfectly ($DoF = 0$, $G^2 = 0$). On the other hand, independence model shows a big variation ($DoF = 7503$, $G^2 = 57$) from saturated model. Removing the 3-way interaction term results in the model $[TP][TU][PU]$ ($DoF = 32$, $G^2 = 22$). Now, we see the effect of removing one 2-way interaction term at a time. Removing term $[PU]$ causes a big deflection from the all 2-way terms model with a small increase in the degrees of freedom ($\triangle G^2 = 7235$, $\triangle DoF = 2$). Removing $[TU]$ also causes some deflection from the all 2-way terms model ($\triangle G^2 = 71, \triangle DoF = 11$); but removing $[TP]$ term causes the smallest deflection and increases the degrees of freedom as well ($\triangle G^2 = 22$, $\triangle DoF = 22$). Further removing terms from $[TU][PU]$ causes greater deflections. The best fit model for a given contingency table is the one with the least $\triangle G^2$ and largest $\triangle DoF$ with respect to the saturated model ($[TU][PU]$ in this case with $\triangle G^2 = 22$ and $\triangle DoF = 22$).

It is clear from finally selected log linear model that there is a dependence between "Transitions" and the "Level of Understanding" as well as between the "Pair Type" and "Level of Understanding".The first dependency is reflected by the term $[TU]$ and the later one is depicted by the term $[PU]$. Where $[PU]$ reflects the fact that a pair of two experts can understand a program better than a pair of two novices. To better understand the dependency between transitions and levels of understanding we use ANOVA. Here, instead of using the transitions, we grouped them in categories as depicted in table 1. Figure 4 shows the differences between the two levels of understanding (medium and high) for the different types of flows in a program.

Table 3: Summary of results of ANOVA for the transitions from the whole interaction

Transition Type	$\mu_{UND=1}(\sigma_{UND=1})$	$\mu_{UND=2}(\sigma_{UND=2})$	F[1,28]	p
Data Flow	0.4 (0.018)	0.45 (0.016)	65.5	$< .01$
Systemaatic Execution	0.55 (0.020)	0.51 (0.017)	32.1	$< .01$

Pairs with medium level of understanding have relatively more transitions amongst all three semantic classes.In terms of gaze transitions this behaviour translates to reading each line of the program and trying to understand it. This shows that these pairs look simultaneously at the conditions in the program as well as the modification of the data elements according to them. They try to understand the data flow in accordance to the control flow of the program. This attempt of program understanding is similar to the "Systematic execution of program". This method is not very characteristic of the pairs with high level of understanding, as shown in an experiment by [10]. The pairs with high level of understanding have relatively more transitions among the identifiers and the expressions. They concentrate more on

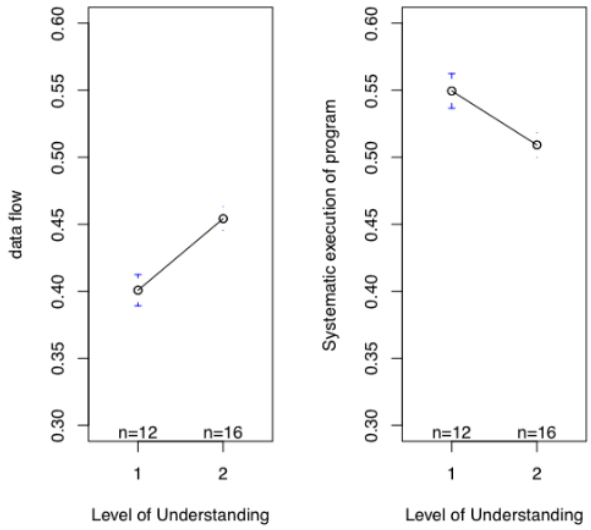

Figure 4: Mean plots for different transitions for the whole interaction (Level of Understanding $1 = Medium$ and $2 = High$)

the variable/entities and the relationship among them. Building up their understanding in this manner the pairs with the higher level of understanding are able to do a proper concept assignment from the program domain to the world domain [21].

5.2 Question 2

Taking our analysis one step ahead, to find the effect of the convergent and divergent episodes of interaction, we carried out 2×2 ANOVA for data flow and data flow according to control flow with two factors level of understanding and convergent/divergent interaction episodes.

Table 4 shows the descriptive statistics for the proportion of data flow transitions in different types of interaction episodes and for different levels of understanding. There were two single effects for the type of interaction episode ($F[2, 28] = 121, p < 0.01$) and for the levels of understanding ($F[2, 28] = 10.86, p < 0.01$), and there was no interaction effect. From figure 5 we see that all the pairs in divergent phases of interaction spend more time on understanding the data flow than that in convergent phases. The effect of the level of understanding on data flow is visible by the fact that the pairs with high level of understanding follow the data flow more than the pairs with medium level of understanding.

Table 4 shows the descriptive statistics for the proportion of systematic execution episodes in different types of interaction episodes and for different levels of understanding. There were two single effects of the type of interaction episode ($F[2, 28] = 106, p < 0.01$) and of the levels of understanding ($F[2, 28] = 8.36, p < 0.01$), and there was no interaction effect. From figure 5 we see that the pairs in convergent phases have a high ratio of transitions that correspond to systematic program execution. There is also an effect of levels of understanding on systematic program execution depicting more effort put by the pairs with medium level of understanding on systematic program execution.

Table 4: Proportions of data flow and systematic execution transitions (mean and standard deviation) by type of episode and level of understanding.

Transition Type	Episode Type	Understanding	
		Low	High
Data Flow	Convergent	0.59 (0.04)	0.56 (0.03)
	Divergent	0.51 (0.02)	0.49 (0.02)
Systematic Execution	Convergent	0.35 (0.04)	0.38 (0.03)
	Divergent	0.44 (0.02)	0.47 (0.02)

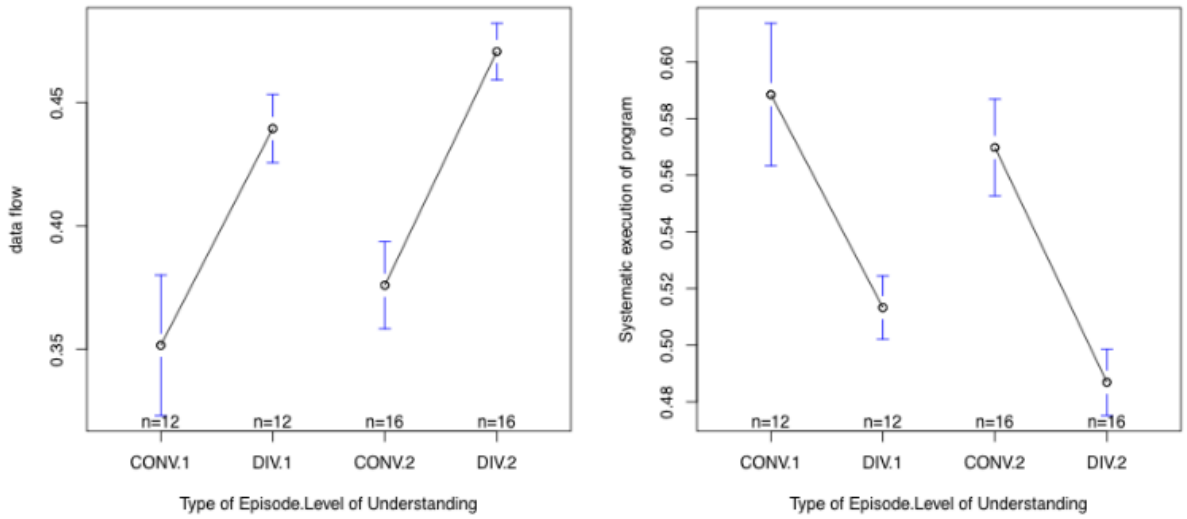

Figure 5: Mean plots for data flow and systematic execution of program for the episodes of interaction and levels of understanding (Level of Understanding $1 = Medium$ and $2 = High$)

6 Discussion and Conclusion

Concerning our first question, we have found evidence for the sensitivity of gaze patterns to the level of understanding. It appears that the gaze of individuals who understood the program better transition more frequently between identifiers and expressions, a transition type that reflects a data flow centred reading of the code. Conversely, individuals with a who got a sense of what the program is doing but were not able to provide the exact explanation, spent relatively more time parsing the program by systematically looking at all types of semantic elements. These findings are compatible with the findings from Jermann and Nüssli (2012) [7] who found that for individual programmers, experts look less than novices at structural elements (type names and keywords) which are not essential when understanding the functionality of the code. Experts look more than novices at the predicates of conditional statements and the expressions (e.g. v /= 10;), which contain the gist of the programs. Our current findings confirm these findings in the context of pairs by using an analysis of gaze transitions between semantic elements . Pairs with high level of understanding put relatively more individual efforts on understanding the entities and their relationships (data flow).

A possible explanation for this difference would be that for the pairs with medium level of understanding some structural elements can act as ”while demons” [3]. On other hand, pairs with high level of understanding show ”as-needed” strategy for building their understanding of the program based on their understanding of the relation between variables in the program [10].

We presented our study for getting the underlying process of the collaborative program comprehension using the eye-tracking data. We put our efforts to distinguish the strategy used for understanding the program by pairs having medium level of understanding from that of pairs having high level of understanding. Pairs with high level of understanding put relatively more individual efforts on understanding the entities and their relationships (data flow). In a convergent episode of interaction, pairs with high level of understanding try to understand the data flow of the program according to the control flow of the program (Systematic program execution). This is attributed to their gaze transitions among all the semantic elements of the program in a convergent phase of interaction and ”to and fro” gaze transitions between expressions and identifiers in the program for the whole interaction (when taken as a whole and in the divergent episodes of interaction).

On other hand, pairs with medium level of understanding put efforts in simultaneous understanding of data and control flow in the program without understanding the meaning of the data variables. This is depicted by their ”back and forth” transitions between expressions and structural elements of the

program while they are in a convergent episode of interaction. This behaviour shows that for the pairs with medium level of understanding some structural elements can act as "while demons" [3]. On other hand, pairs with high level of understanding show "as-needed" strategy for building their understanding of the program based on their understanding of the relation between variables in the program [10].

Concerning our second question, we have shown that in convergent episodes of interaction, pairs with high level of understanding as well as pairs with medium level of understanding try to understand the program via a strategy of systematic program execution. This is depicted by their "back and forth" transitions between expressions and structural elements of the program. In comparison, the data flow transitions are less frequent in divergent episodes.

A possible explanation for the differences between convergent and divergent episodes is that programmers are visually searching the code for variable and method names during the divergent phases and that in this case the augmentation of data flow transitions stems from a selective exploration of the code. Another explanation is that during divergent episodes, programmers focus on building basic knowledge about variables and expression which is then discussed during convergent episodes where structural elements of the code are used to define the joint focus of attention. An analysis of the dialogue between partners will help to understand these subtle differences.

Since there were no interaction effects between these factors, we can conclude that both the level of understanding and the type of episode affect the types of gaze transitions that are observed. It is striking that the differences between convergent and divergent episodes are twice as large as the differences between levels of understanding. This seems to indicate that gaze indicators are more sensitive to task-related aspects than to levels of expertise.

References

1. J.R. Anderson. *Cognitive Psychology and its Implications*. Worth Publishers, 1985.
2. P. Baheti and L. Williams. Exploring pair programming in distributed object-oriented team projects. In *In Proceedings of XP/Agile Universe 2002*. Springer Verlag, 2002.
3. J. Bonar and E. Soloway. Uncovering principles of novice programming. In *Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '83, 1983.
4. R. Brooks. Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, 18(6), 1983.
5. R. Dale D. C. Richardson and N. Z. Kirkham. The art of conversation is coordination. *Psychological Science*, 18(5):407–413, 2007.
6. J. M. Gottman and A. K. Roy. *Sequential Analysis - A Guide for Behavioral Researchers*. Cambridge University Press.
7. P. Jermann and M.-A. Nussli. Effects of sharing text selections on gaze cross-recurrence and interaction quality in a pair programming task. In *In Proceedings of Computer Supported Collaborative Work 2012*, 2012.
8. W.L. Johnson and E. Soloway. Proust: Knowledge-based program understanding. *Software Engineering, IEEE Transactions on*, SE-11(3), 1985.
9. Y. Kawahara. Change-point detection in time-series data by direct density-ratio estimation. *Direct*, 4(2), 2009.
10. J. Koenemann and S.P. Robertson. Expert problem solving strategies for program comprehension. In *CHI '91 Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, 1991.
11. J. Larkin. *Cognitive Skills and Their Acquisition*, chapter Enriching Formal Knowledge: A model for learning to solve textbook physics problems. Lawrence Erlbaum Associates, 1981.
12. S. L. Lauritzen. *Lectures on Contingency Tables*. University of Aalborg, 1989.
13. S. Letovsky. Cognitive processes in program comprehension. *Journal of Systems and Software*, 7(4), 1987.
14. A. Von Mayrhauser and A.M.Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8), 1995.
15. M.-A. Nussli. *Dual-Eye Tracking Methods for the Study of Remote Collaborative Problem Solving*. PhD thesis, Ecole Polytechnique Federale de Lausanne, 2011.
16. R. P. Adams and D. J. C. MacKay. Bayesian Online Changepoint Detection. *ArXiv e-prints*, October 2007.
17. D. C. Richardson and R. Dale. Looking to understand: The coupling between speakers' and listeners' eye movements and its relationship to discourse comprehension. *Cognitive Science*, 29(6):1045–1060, 2005.
18. B. Shneiderman and R. Mayer. Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International Journal of Parallel Programming*, 8, 1979. 10.1007/BF00977789.
19. E. Soloway and K. Ehrlich. Empirical studies of programming knowledge. *Software Engineering, IEEE Transactions on*, SE-10(5), 1984.
20. S. Paul S.R. Tilley and D.B. Smith. Towards a framework for program understanding. In *Program Comprehension, 1996, Proceedings., Fourth Workshop on*.
21. B. G. Mitbander T. J. Biggerstaff and D. E. Webster. Program understanding and the concept assignment problem. *Commun. ACM*, 37(5).
22. E. Terzi and et al. Efficient algorithms for sequence segmentation.