Collaboration in Pair Programming: Driving and Switching

Laura Plonka, Judith Segal, Helen Sharp, Janet van der Linden

Centre for Research in Computing, The Open University, Milton Keynes, UK,
http://crc.open.ac.uk/

Abstract. This paper reports on an empirical study about the mechanisms of the collaboration of drivers and navigators in Pair Programming (PP) sessions. Based on video recordings of professional software developers, we analysed the mechanisms of role switches and how developers split the task of driving. We found that developers do not evenly contribute to the task of driving and that they spend on average a third of the session without any computer interaction focusing mainly on communication. In addition, our results show that most pairs switch roles frequently and that the frequency and fluidity of switches indicate a high level of engagement on the part of both developers.

Key words: Pair programming, empirical studies, collaboration

1 Introduction

Pair Programming (PP) is a software development practice in which two programmers are working together on one computer, sharing mouse and keyboard [2, 17]. Over recent years, a wide range of studies [3, 13, 15, 16, 18] has investigated PP; most of those studies report positive results indicating benefits of PP. However, some studies are inconclusive and some studies even report contradictory results (see [10]). This accentuates the need for further studies. External factors have been identified that influence the effects of PP, e.g. task complexity [1] and personality [11], but there are still open questions regarding the effect of the developers' behaviour within PP sessions.

There are two roles in each pair: driver and navigator. The programmer who is typing is typically called the "driver"and the other one is called the "navigator". The programmers can switch these roles at any time. In this paper, we analyse some aspects of the role behaviour of driver and navigator such as the role distribution, the role switches and the mechanisms of switching based on PP sessions in industrial settings. The results provide new insights into the collaboration of the developers in terms of how developers split the task of "driving" and how and by whom switches are initiated.

2 Related Work: The Roles of Driver and Navigator

Originally, the literature claimed that driver and navigator work on different levels of abstraction; the navigator acts and thinks on a strategic level [2, 8] , is responsible for reviewing the driver's work [8, 12], and for "watching for

defects, thinking of alternatives, looking up resources" [18] whereas the driver is responsible "for typing the code" [8] and for finding "the best way to implement this method right here" [2].

However, these role descriptions have been challenged by the results of recent studies [6, 7, 9] that show that driver and navigator do not think or act on different levels of abstraction. Freudenberg et al. [6, 9] conducted four one-week studies with experienced pair programmers in industrial settings. They showed firstly that a significant amount of talk is at an intermediate level of abstraction and secondly that driver and navigator tend to work on the same level of abstraction. The second result is also reported by Chong and Hurlbutt [7]. They stated that "aside from the task of typing" there was no division of labour between the driver and navigator. Furthermore, Freudenberg et al. reported [9] that driver and navigator switch roles regularly and with a fluid handover during a PP session and conclude that this implies that the navigator continually maintains a good understanding at different levels of abstraction. They suggest defining the roles of driver and navigator by "the additional physical and cognitive load of typing borne by the driver" rather than "by segmenting the problem space according to level of abstraction".

Another aspect of the roles of driver and navigator are the role switches. Based on the experience in one XP team , Dick and Zarnett [8] reported that developers did not switch roles and that this led to a drift of the navigator's attention. As a result, the pair did not develop a simple design. In addition, they observed a lack of communication between the driver and the navigator and reported that this resulted in substandard design. Vanhanen et al. [16] conducted a single case study in a large telecommunication company. They found that developers switched roles 2-3 times a day and reported that those switches typically took place after the lunch break or after a personal break of the driver. Additionally, they stated that developers who were already familiar with the task act as a driver and those who did not know the code felt that it was impossible to act as a driver. In contrast to Vanhanen et al., Rostaher and Hericko [14] found that developers switched roles on average 21 times in an average period of 358 min. Those results are based on self-reported data by professional developers; developers were asked to keep track of their switches using a software system. Additionally, they found that the frequency of changes correlated with programming experience and that more experienced pairs changed most often.

A more detailed analysis of the handover process for switching roles and the effects of switching and driving was conducted by Freudenberg et al. [5] and by Chong and Hurlbutt [7]. Freudenberg et al. observed 36 PP sessions (18 different pair constellations) all of which were audio taped and three of which were video recorded. All pairs worked on one computer using one mouse and one keyboard. For their study, they defined "driver" by saying that "the possession of the keyboard signalled who is in the driver role". They found that switches were mostly initiated by the driver; the keyboard was "offered and accepted" rather than being "taken without offering" [5]. Furthermore, they pointed out that mouse/keyboard were used both as input devices and as pointers to highlight

the object of conversation on the screen. In addition, Freudenberg et al. [4] investigated developer collaboration; they found that PP is highly collaborative and that driver and navigator contribute almost equally to every subtask.

Chong and Hurlbutt [7] observed two different development teams. They sat behind the developers and took notes. Some of the sessions were audio-recorded. The data was analysed using a self-developed coding scheme. Chong and Hurlbutt analysed the keyboard and mouse switching and the effect of controlling the keyboard. They found that pairs who work on a shared machine and use dual mice/keyboards switch more frequently than pairs with a single mouse/keyboard; developers using dual mice/keyboards did not drive at the same time but the navigator jumped in during pauses or periods of hesitation. According to Chong and Hurlbutt, the switches occurred when "it was easier for a programmer to execute an action himself" when a developer "was more practiced in a particular subtask", or when one developer was called away during the session. Additionally, it seemed that sometimes developers were just eager to type. As an effect of switching, Chong and Hurlbutt stated that frequent switches helped the developers to "maintain a high level of mutual awareness of each other's action". In contrast, where developer pairs used a single mouse and a single keyboard, "switching required more coordination and an explicit physical relocation of the keyboard", the developer with control over the keyboard in the beginning "generally retained control for the whole session" and "maintaining active engagement in the task appeared more effortful". Additionally, they found that being driver affects the power in decision-making.

These studies provide insight into the effects of switching and driving, which indicate that the number of switches and the driving distribution affect PP and should be investigated in more detail. However, none of the studies above focuses on driving times or how driving times are distributed among the developers, and most analysis about switches is based on contemporaneous notes and audio-recordings rather than video recordings (except for three sessions in [5]). Since the switches include non verbal interactions such as shifting the keyboard and those switches can happen very frequently [5, 7], we analyse switches based on video recordings. This allows us to replay important episodes and to analyse switches in more detail. Furthermore, video recordings allow us to measure the driving times of the developers.

3 Study Methodology

This paper is based on four one-week studies in four different companies. During our studies, the developers worked on their day-to-day tasks in their usual working environment. In this section, the study background, data gathering and analysis are described.

3.1 Data gathering

Audio and video recordings were taken of the PP sessions. Using these recordings the driving times of each developer, the non–driving times (episodes in which neither developer is driving), and the switching behaviour were analysed, along

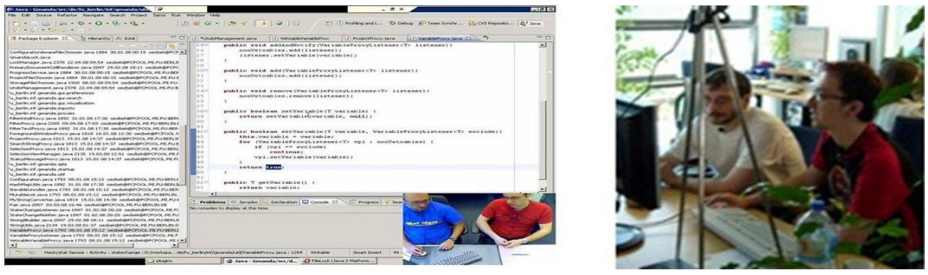

Fig. 1. On the left: Screenshot of a fully synchronized video of a PP session: This shows the developers' screen (Eclipse IDE) and, bottom right, the video recording of the developers. On the right: Recording setup in one of the companies. The setup was installed in the usual working environment of the developers.

with the communication of the developers.

The recordings of the session consist of the following three data sources:

- Audio recordings of the verbal communication between the participants,
- a video of the programmers capturing who is driver and who is navigator,
- and a full-resolution screen recording capturing all computer activities.

All three data sources are fully synchronized and stored in a single video file so that all information is available at once (see figure 1). The PP sessions were recorded in the developers' day-to-day work environment and while solving day-to-day tasks. Our recording setup was installed at one computer in each participating company (see figure 1). The location of this computer was chosen by the company. In one company the recording equipment was at a computer located in an extra room, in the three other companies it was in the development team office. Although the developer pairs had to work physically on this computer for the recordings they were able to work virtually at one of the developers' machines using a remote desktop setup.

3.2 Data Background

We recorded 21 PP session in four different companies. All companies work in different industries and all used agile approaches in the teams we observed. Two of the companies had just introduced PP and used Scrum, the other two had used PP for more than one year and used an adapted agile approach. In all four companies, PP was encouraged by the team leaders but the team leaders did not assign pairs or tasks. Instead the developers decided when to use PP and in which pair constellation. The forming of the pairs was done during their daily meeting or spontaneously during the day.

Participation in the study was voluntary for all developers; in total 20 different pair constellations with 31 developers were recorded, with some developers being recorded more than once. 17 pairs used one mouse/keyboard and four pairs used dual keyboards and mice (the developers decided this). A PP session lasted between one and a half hours to three and a half hours; in total we video taped about 37 hours of PP sessions. Table 1 provides an overview of the recordings.

Industry	# of recorded sessions	# of participating developers	# of different pair constellations	av programming experience (in years)	av PP experience (in years)
Geographic information systems	6	8	6	7.9	6.5
Transport and logistics	7	8	6	9.7	1
Email marketing	4	6	4	7.1	2.9
CRM systems estate	4	7	4	6.8	1.6

Table 1. Overview: Number of recordings and developers' backgrounds

3.3 Analysis process

The video data was coded based on a self-developed coding scheme and subsequently the occurrence of some of the codes was counted, for example, to investigate the frequency of switches; or the length of time of a code was recorded, for example to measure driver distribution among the developers.

Coding scheme driving and non–driving times	
Code	**Description**
D1.isDriving, D2.isDriving (D1: Developer 1, D2: Developer 2) (driving time)	Driving is coded when a developer is using the mouse or the keyboard. Having a hand on the mouse or keyboard without using it is not coded as driving.
BothDriving (driving time)	BothDriving is coded when both developers have their hands on mouse and keyboard, the keyboard/mouse is in use but it is not apparent who is using it.
NoOneIsDriving (Non–driving time)	NoOneIsDriving is coded if none of the developers are driving, e.g. when they are discussing something.

Table 2. Coding scheme for analysing non–driving and driving times

Definition of the term of driving

In the context of this study, we define "driving" as being in control over mouse/keyboard and using one or both of them as an input device (this can include browsing, writing code, executing programs, etc.). This means that it is not considered as driving when developers only have their hand on mouse/keyboard. This definition evolved during the analysis based on the observation that some developers tend to place their hand on the keyboard or mouse although they are not typing; for example, when they are engaged in a discussion with their partner.

We will refer to the developer who is driving as the driver and to the other as the navigator. The developers keep those role names (even when the driver stops typing) until they switch roles. For this paper navigator is defined as "not being

the driver" without implying any kind of behaviour in the role of the navigator.

Coding scheme switches: Who initiates switches?		
Code	**Description**	**Examples**
Driver-Initiated	A switch is initiated by the driver if the driver offers the keyboard/mouse to the navigator. This can happen – by passing the keyboard (physically), – by offering the keyboard/mouse verbally to the navigator, – or by a combination of both. Additionally, a switch is driver initiated if the driver states that he does not know how to continue and pushes the keyboard away from himself (even if he does not pass it to the navigator).	1.Example: Driver: *"Now it's your turn. I think it makes sense (if you drive now)."* Navigator: *"Yes, it's my turn now because we want to do the java debugging."* 2.Example: Driver: *"I do not know how."* Driver takes his hands from the keyboard Navigator: *"Ok."* Navigator takes the keyboard
Navigator-Initiated	A switch initiated by the navigator if – the navigator asks for the keyboard/mouse (verbally), – or the navigator just takes the keyboard/mouse (physically).	1.Example: Navigator: *"May I?"* 2.Example: Navigator: *"It's too complicated to dictate, let me do it."*
Undefined	A switch is initiated by "undefined" if right before the switch both developers had their hand on the keyboard and mouse at the same time (one developer is using the mouse and the other one the keyboard).	-
D1Initiated, D2Initiated	For each switch it was coded which developer initiated the switch.	-

Table 3. Coding scheme for analysing switches: Who initiates the switches?

Definition of the term of switching

We define switches as a change of driver. This means a switch occurs whenever a navigator becomes a driver. Developers can switch roles without any driving breaks or developers can switch after a driving break; for example, one developer is driving, then the developers start a discussion and after this discussion the other developer becomes the driver. However, it is not a switch if the same developer continues driving after the discussion.

Coding scheme for analysing driving and non–driving times

The driving and non–driving times are coded in the videos and afterwards quantitatively analysed to investigate the percentage of driving and non–driving times of the PP session. Certain events and episodes during the session can greatly influence the percentage of the non–driving times and the driving times of each developer. Those events include stand up meetings that might take place during the recordings (this would increase the non–driving time), breaks of both developers or the absence of one developer (the other developer might continue

driving during this time, so it might affect the driving distributions among the developers). To reduce bias, episodes in which one or both of the developers were absent during the session were cut out and not included in the calculation of driving and non–driving times. Table 2 describes the "driving" codes.

Coding scheme for analysing switching behaviour

A switch is a change of driver. A switch can be initiated verbally or physically by the navigator or the driver. For analysing the switches we developed a coding scheme that focuses on who is initiating a switch (by role (driver or navigator) and by developer) and how a switch is initiated. Table 3 provides an overview of the codes that were used to identify who initiated the switches. Table 4 presents the coding scheme to investigate how developers switch roles.

Fig. 2. Example: Switch physically initiated by the driver
Context: In this phase of the session, the developers switch roles according to small subtasks. In this example, the developer on the left just finished a simple subtask and passes the keyboard without any comment -just a smile on his face- to his partner. His partner says: *"Ah, you take the easy road again!"* and takes the keyboard. The developer on the left starts laughing.

Fig. 3. Example: Switch physically initiated by navigator
This example shows how the navigator (on the left) initiates a switch although the mouse is still being used by the driver. The navigator positions his hand (left hand) on the mouse without asking and the driver takes his hand away. The developer on the left passes the mouse from his left to his right hand and starts driving.

4 Results

In this section, we firstly describe the results of the driving and non–driving times and then the role switches. The implications of our results will be discussed in section 5.

4.1 Driving and non–driving times

For each PP session, the length of driving times and non–driving times were analysed. We found that PP sessions consist on average of 33.3% of non–driving

Coding scheme switches: How are switches initiated?		
Code	**Description**	**Examples**
physically-Initiated	A switch is initiated physically when – the driver passes the keyboard to the navigator. – the navigator takes the keyboard without asking although it wasn't offered by the driver.	For example, see figures 3 and 2.
verballyInitiated	A switch is initiated verbally when – the navigator asks the driver to switch, – the driver verbally offers the keyboard/mouse to the navigator or asks the navigator to switch.	1.Example: Driver: *"It's about the coupon system. So it's your turn now."* 2.Example: Driver: *"Do you want to do it?"*
verballyAnd-PhysicallyInitiated	A switch is initiated verbally and physically when – the driver asks the navigator to switch and is passing the keyboard at the same time, – the driver states that s/he does not know how to continue and pushes the keyboard away, – when the navigator asks to switch and takes the keyboard while asking without waiting for any response of the driver.	Example: Navigator: *"May I suggest something?"* Navigator is taking the keyboard while saying that.
Conflict	A conflict occurs if after a non–driving time, both developers try to drive at the same time.	-
Reject	Reject is coded when a developer attempts to switch but the partner refuses to switch.	Example: Driver: *"Do you want to continue?"* Nav: *"I think you know it better. I'm, hm, we reached the limit of my knowledge."* Driver continues driving.

Table 4. Coding scheme for analysing switches: How are switches initiated?

times and 66.7% of driving times (the sum of the driving times of both developers).

Break of driving or non–driving time?
The driving and non–driving times were coded in the videos; sequences were coded as driving when a developer was typing or using the mouse and as soon as they stopped typing or stopped using the mouse the sequence was coded as non–driving. This means our coding does not distinguish between a short driving break (the driver stops typing just for a second and then starts again) and non–driving times. We decided to investigate how those short driving breaks would affect the non–driving times. We found that the percentage of non–driving times would decrease by 2.4% percent if we would consider breaks up to 3 seconds not

as non–driving times and by 4.1% for breaks up to 5 seconds. This shows that those small breaks do not highly affect the percentage of non–driving times.

4.2 Non–driving times

Given that the proportion of non–driving times was relatively high, we decided to investigate the non–driving times in detail. Figure 4 provides a graphical representation (box plot) of the non–driving times. This figure shows that in half of the sessions the percentage of the non–driving times lies between 27.8% and 36.8% with a median of 31.2% and it shows four outliers in the data set; one session has a very low amount and three sessions have a very high amount of non–driving times (up to 54.5% of the session). The fact that developers spend on average a third of the session without any computer interaction led us to investigate the reasons for these non–driving times.

Reasons for non–driving times
Based on the video analysis, we found five reasons for non–driving times.

- **Waiting times**
  There are times that developers have to wait for the computer to be responsive again, for example when compiling code, running tests or starting applications. Additionally, developers sometimes face technical problems, for example, problems with version control systems or problems with the IDE that make it unresponsive for a while.
- **Discussions/Explanations**
  Most pairs seem to have a continuous flow of communication during the whole session. However, there are episodes during the session in which the developers concentrate exclusively on discussions and explanations. While focusing on the explanations, it seemed that the non–driving times were sometimes interrupted by very short sequences of driving times in which developers use mouse and cursor as a pointer to highlight certain lines on the screen to support their explanations. This means they explain, point, and explain again.
- **Use of external representations**
  Some developers seem to prefer to discuss the structure of the code/design and their solution strategies on paper. Hence, the developers use a sheet of paper to draw, explain and discuss their ideas without using the computer.
- **Searching for advice from a third party**
  Pair programmers can use each other as an information resource. Nevertheless, there are situations in which the expertise of the two developers is not enough to make a decision, or situations in which a developer pair cannot solve the problem. In those cases developers can involve other developers or the team leader in their discussion in order to find a solution and progress with the task. During those times both developers usually stop typing and concentrate on the conversation.
- **Interruptions**
  Pair programmers get interrupted by other developers or even by co-workers from different departments. While in some cases one developer deals with the interruption and the other one continues the work, in other cases both

developers stop working in order to deal with the interruption. This leads to non–driving times for the pair.

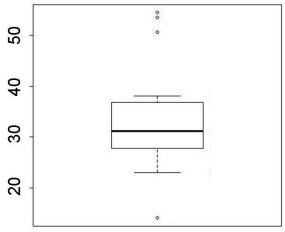

Fig. 4. Box plot: Percent of non–driving times per session from 21 PP sessions

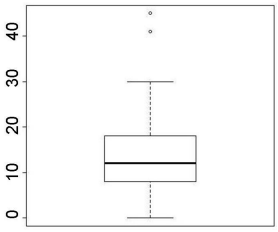

Fig. 5. Box plot: Average number of switches per hour for all 21 sessions

The proportion for each reason hasn't yet been analysed in detail, but based on our observation of waiting times, discussion and explanation and the use of external representations seem to be the major reasons for the non–driving times. The pairs of the three outlier sessions with a very high amount of non–driving time faced serious technical problems or had difficulties finding an approach to solve the task or were working on a very complex problem which led to the use of external representations and discussions. The low amount of non–driving times in one outlier session can be explained by the fact that one developer was familiar with the task and knew a strategy how to solve the task already.

4.3 Driving times: Driver distribution among the developers

In this section, the analysis focuses on the driver distribution among the developers to investigate if the driver distribution is balanced among the developers.

Therefore, we analysed the percentage of driving time for each developer with respect to the total driving time of each session (so non–driving times are not included). We say the driver distribution is balanced if the driving times of each developer is between 40% and 60%. We found that the driver distribution is balanced in 7 out of 21 sessions whereas the distribution is extremely unbalanced in four sessions.

4.4 Role switches

We coded the role switches according to the coding scheme presented in table 3 and table 4. In total, we analysed 512 role switches in the 21 sessions. The codes "Reject" and "Conflict" hardly occurred (on average less then once per hour) and hence, we will not focus on them in detail.

How often do pair programmers switch roles?

As a result of our analysis, we found that developers switch roles on average 15.67 times per hour with a standard deviation of 12.19. Due to this high standard deviation, the number of average role switches will be presented as a box plot.

Figure 5 shows that in half of the sessions the number of switches lies between 8 and 18 with a median of 12 switches per hour. Additionally, it shows two outliers with more than 40 switches per hour.

Are switches initiated evenly by both developers?
Based on the results of the driving times, we found that in two thirds of the cases one developer dominates the driving. We investigated whether the switches are initiated by both developers or whether one developer dominates the initiation of switches. In 16 sessions, the initiation of switches is balanced among the developers (each developer initiates 40 to 60 percent of the switches). In four sessions, one developer dominates the initiation of switches and in one session the developers do not switch at all. This shows that even though the driver distribution is mostly dominated by one developer, the switches are in most cases evenly initiated by both developers.

Correlation between driver distribution and number of switches
We analysed the correlation between the number of switches and the driving distribution among the developers. The pairs who had a balanced driver distribution switched roles on average 16 times per hour with a standard deviation of 10.12. The pairs with an unbalanced driver distribution had on average 16.46 role switches per hour with a standard deviation of 13.73. This indicates that the number of switches does not correlate with the driver distribution among the developers. The driver distribution can be independent of the number of switches because developers can switch just to show or to point at something on the screen and then switch back. Figure 6 visualizes this pattern of switches.

Fig. 6. This figure illustrates that the driver distribution can be independent of the number of switches. In this example, developer 2 is driving only for a short period of time and developer 1 is driving for longer periods. Even though this pair might switch on a regular basis, their driver distribution is not necessarily balanced.

Which role initiates switches?
Additionally, we investigated the role of the switch initiator. Switches are initiated by the navigator in 82% of the cases and in 14% by the driver. In the remaining 4%, the initiation could not be clearly assigned (see figure 7).

What are the mechanisms for switching roles?
Developers can initiate switches by asking or offering the keyboard/mouse verbally or they can just pass or take it. We analysed which mechanisms they use to switch roles. We found that developers initiate switches physically without any verbal cues in 81% of all switches. However, before they take the mouse or keyboard there are usually non verbal cues as for example leaning forward or placing their hands next to the mouse or the keyboard before they actually take it. In 11% of the switches the developers initiated a switch verbally and

in 8% developers combined verbal and non verbal cues to initiate a switch (see figure 8).

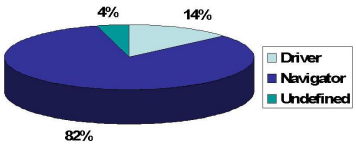

Fig. 7. Which role initiates switches?

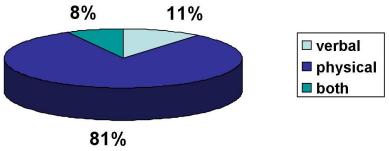

Fig. 8. How are the switches initiated?

5 Discussion

5.1 Discussion of the results

Here we discuss our results presented above, which we summarize as:

1. A PP session consists of on average 33.3% non–driving times.
2. Developers do not split the driving times evenly; instead in two thirds of the cases one developer dominates the driving.
3. Developers switch roles on average 15.67 times per hour.
4. In most of the sessions both developers (not roles) initiate switches evenly.
5. Most of the switches (81%) are physically initiated.
6. 82% of the switches are initiated by the navigator.
7. The codes "reject" and "conflict" hardly occurred (on average less than once per hour).

These results lead us to the following conclusions:

Communication intensive non–driving times

Our first result shows that developers spend on average a third of a PP session without any computer interactions. We found five reasons for these times: (a) waiting time, (b) explanation/discussions, (c) the use of external representations, and to a lesser degree (d) searching for advice from a third party, and (e) interruptions. Waiting times (a) and interruptions (e) are not initiated by the developers. We haven't analysed waiting times in detail, but we assume that developers continue their usual communication during those times. In the cases (b)-(d), the non–driving times are initiated by the developers and lead to communication intensive activities which are focused on the current task. Searching for advice requires communication with a third party, whereas explanation/discussions and the use of external representations lead to intra pair communication. During these times developers stop their computer interaction in order to focus exclusively on communication.

The relatively high amount of non–driving times and the communication intensive activities which take place during these times emphasize the role of communication among the developers in PP sessions.

Unbalanced driving distribution and drivers don't mind driving
We found (result 2) that in most cases developers do not split the driving times evenly: usually one developer dominates the driving so developers do not contribute evenly to the task in terms of driving. However, the driver doesn't seem to initiate switches often (result 7: only 14% of the switches are initiated by the driver) in order to balance the driving distribution among the developers. This indicates that an unbalanced distribution does not seem to be an issue for the dominating driver and that the driver does not perceive typing as additional workload.

Freudenberg et al. [9] suggest that the task of driving is an "additional physical and cognitive load" for the driver. Our results challenge the suggestion that the driver perceives driving as a "load" as we did not find any indicator for that.

Simple and intuitive handover process
Result 5 shows that 81% of the switches are physically initiated without any verbal communication (only with non verbal cues). The physically initiated switches indicate that switching roles is a very fluid and effortless process that does not require a complicated handover mechanism. This is also supported by the fact (result 7) that we hardly found any "rejects" or "conflicts" in the sessions.

The finding about the simple and intuitive handover process supports the results by Freudenberg et al. [9] that switching is a very fluid process. However, the fact that switches are mostly initiated by the navigator (result 6) contradicts the results reported by Freudenberg et al. [5] who reported that most of the switches (switch of the keyboard) were initiated by the driver.

Driver and navigator focus on the same subtask
Most of the switches (82%) are initiated by the navigator (result 6). Even in these cases, the process of switching is very effortless and does not require any kind of extra explanation from the driver. For example, the driver does not explain the next steps before handing the keyboard and mouse over to his partner. This indicates that the navigator was actively involved and understood the work of the driver as this is a precondition to take over control so effortlessly.

This is consistent with the findings from Freudenberg et al. [6, 9] and Chong [7] who stated that driver and navigator work on the same level of abstraction and switch between these levels as a pair.

Switches as an indicator for engagement
We found that pairs switch on average 15.67 times per hour (result 3) and that in most PP sessions switches are initiated by both developers (result 4). So, both developers use the mechanism of switching regularly. A high frequency of switching could indicate a good engagement of both developers because initiating a switch and continuing to drive (even for a short period of time) requires an understanding of the current activity. Hence, frequent switches might be an indicator for both developers being actively involved during the whole PP session. However, the reverse (few switches means little engagement) can't be concluded based on our data.

Interestingly, the driving distribution can't be used as an indicator for engagement; even a balanced driving distribution does not mean that both developers

are actively involved during the session, for example, one developer is driving the first half of the session the other one the second half, but they may not be engaged while their partner is driving.

The finding that switches could be an indicator for engagement is consistent with Chong and Hurlbutt's finding [7] that frequent switches help to maintain mutual awareness of each other's action. Additionally, Chong and Hurlbutt [7] and Dick and Zarnet [8] stated that a lack of switching has a negative effect such as a drift of the navigator's attention and that "maintaining active engagement appeared more effortful". However, the frequency of switching in our study is higher than in the studies conducted by Dick and Zarnet [8] and Vanhanen et al. [16] who reported that pairs switched not at all or 2-3 times a day, respectively.

Some of our results and conclusions contradict results from existing literature, for example, the frequency of switches. This might be due to a different data gathering and analysis process. Our study focuses mainly on the switching and driving behaviour of the developers, therefore we gathered very detailed data using video recordings of the PP sessions and analysed the data by coding these videos in a way that the results can be quantified as well. Most of the other studies [6, 7, 8, 9, 14] used observational data, audio recordings or self-reported data. Those data sources capture less detailed data about the switching and driving behaviour. However, the main focus of those studies was different and the data served their purpose. Additionally, some of these studies might have used a different definition of switching and driving and some papers do not define these terms. These differences in the studies could lead to different results.

5.2 Limitations of the study

Our results are based on PP sessions from four different companies. Although these companies work in different industries, generalisability might be limited by the small number of companies and the fact that all companies were based in Germany. Participation in the study was voluntary, hence, it is possible that developers with a positive attitude towards PP or who are confident about their PP skills are more likely to take part. Nevertheless, our participants had a range of programming and PP experience. Finally, the possible effect of being video taped should be taken into account before generalising the results.

6 Future work

This study provides new findings about driving and switching behaviours of pair programmers. We have shown how developers split the driving time, how they spend the non–driving times, and how and how often they switch roles. However, not all developer pairs have a similar amount of non–driving times, a similar driving distribution, or the same frequency of switches. This raises the questions of how developers decide who is driving and why developers initiate switches. In addition, we found that the non–driving times are communication intensive and argued that the frequency of switches could be an indicator of the engagement of the developers. We plan to investigate the effect of non–driving times, the driving distribution and the switches on the engagement and collaboration of

the developers further. In our next study, for which we have already gathered different kinds of data, we will address the following questions:

1. How does the amount of non–driving time affect communication in PP? Are non–driving times more communication-intensive than driving times?
2. What factors (for example, PP experience, hardware setting) influence the driving distribution? Does the driving distribution affect the collaboration of the developers?
3. Why do developers switch roles? How does the frequency of switches affect the collaboration of the developers?

The answers to these questions will provide us with a rich picture about the roles of driver and navigator.

7 Summary

This paper presents new insights into switching and driving behaviours of developers and the amount of non–driving times together with an analysis of the reasons for those times based on a detailed video analysis of industrial developers. We provide a new perspective on the collaboration of the developers: we have shown that developers do not evenly contribute to the task of driving, that the driving distribution cannot be used as an indicator for the engagement of the developers, but that a high frequency of switches might indicate that both developers are actively engaged throughout the PP session. Additionally, we found that independent of the driving times, most pairs switch roles frequently and in a very fluid way. The role switches are mostly physically initiated by the navigator without any complicated handover process or additional explanation by the driver. This simple handover process indicates that both developers know the current activity. The fact that most of the switches are not initiated by the driver suggests that the driver does not perceive the task of driving as an additional workload as suggested in the literature.

Moreover, we found that developers spend on average one third of the session without any computer interactions and that these non–driving times are communication intensive. The relatively high amount of non–driving time emphasizes the crucial role of communication in PP. Our future work includes investigating the effects of the switching and driving behaviour on the communication between the developers and additionally identifying which factors, for example, the PP experience of the developers or the hardware setting (dual mice/keyboard vs. single mouse/keyboard) influence those behaviours.

8 Acknowledgements

The authors would like to thank the participating companies.

References

[1] Erik Arisholm, Hans Gallis, Tore Dyba, and Dag I.K. Sjoberg. Evaluating pair programming with respect to system complexity and programmer expertise. *IEEE Trans. Softw. Eng.*, 33:65–86, February 2007.

[2] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 1999.
[3] Andrew Begel and Nachiappan Nagappan. Pair programming: what's in it for me? In *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 120–128. ACM, 2008.
[4] S. Bryant, P. Romero, and B. Boulay. The collaborative nature of pair programming. *Lecture Notes in computer Science*, 4044:53, 2006.
[5] S. Bryant, P. Romero, and B. du Boulay. Pair programming and the reappropriation of individual tools for collaborative programming. In *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, pages 332–333. ACM New York, NY, USA, 2005.
[6] S. Bryant, P. Romero, and B. du Boulay. Pair programming and the mysterious role of the navigator. *International Journal of Human Computer Studies*, 66(7):519–529, 2008.
[7] J. Chong and T. Hurlbutt. The social dynamics of pair programming. In *Proceedings of ICSE 2007*, pages 354–363, 2007.
[8] A.J. Dick and B. Zarnett. Paired programming and personality traits. In *Proceedings of XP2002*, 2002.
[9] S. Freudenberg, P. Romero, and B. du Boulay. "talking the talk": Is intermediate-level conversation the key to the pair programming success story? *AGILE 2007*, pages 84–91, 2007.
[10] J.E. Hannay, T. Dybå, E. Arisholm, and D.I.K. Sjøberg. The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, 51(7):1110–1122, 2009.
[11] Jo E. Hannay, Erik Arisholm, Harald Engvik, and Dag I.K. Sjoberg. Effects of personality on pair programming. *IEEE Transactions on Software Engineering*, 36:61–80, 2010.
[12] R. Jensen. A pair programming experience. *The journal of defensive software engineering*, pages 22–24, 2003.
[13] L. Layman, L. Williams, and L. Cunningham. Exploring extreme programming in context: an industrial case study. In *Agile Development Conference, 2004*, pages 32–41, 2004.
[14] M. Rostaher and M. Hericko. Tracking test first pair programming-an experiment. *Lecture notes in computer science*, pages 174–184, 2002.
[15] J. Vanhanen and C. Lassenius. Perceived effects of pair programming in an industrial context. In *Software Engineering and Advanced Applications, 2007. 33rd EUROMICRO Conference on*, pages 211–218, 2007.
[16] Jari Vanhanen and Harri Korpi. Experiences of using pair programming in an agile project. In *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, page 274b. IEEE Computer Society, 2007.
[17] L. Williams and R. Kessler. *Pair programming illuminated*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2002.
[18] L. Williams, RR Kessler, W. Cunningham, and R. Jeffries. Strengthening the case for pair programming. *IEEE software*, 17(4):19–25, 2000.