

# Open Research Online

---

The Open University's repository of research publications and other research outputs

## Knowledge transfer in pair programming: an in-depth analysis

### Journal Item

#### How to cite:

Plonka, Laura; Sharp, Helen; van der Linden, Janet and Dittrich, Yvonne (2015). Knowledge transfer in pair programming: an in-depth analysis. *International Journal of Human-Computer Studies*, 73(1) pp. 66–78.

For guidance on citations see [FAQs](#).

© 2014 Elsevier Ltd



<https://creativecommons.org/licenses/by-nc-nd/4.0/>

Version: [not recorded]

Link(s) to article on publisher's website:

<http://dx.doi.org/doi:10.1016/j.ijhcs.2014.09.001>

---

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online's data [policy](#) on reuse of materials please consult the policies page.

---

[oro.open.ac.uk](http://oro.open.ac.uk)

# Open Research Online

---

The Open University's repository of research publications and other research outputs

## Knowledge transfer in pair programming: an in-depth analysis

### Journal Item

#### How to cite:

Plonka, Laura; Sharp, Helen; van der Linden, Janet and Dittrich, Yvonne (2015). Knowledge transfer in pair programming: an in-depth analysis. *International Journal of Human-Computer Studies*, 73(1) pp. 66–78.

For guidance on citations see [FAQs](#).

© 2014 Elsevier Ltd



<https://creativecommons.org/licenses/by-nc-nd/4.0/>

Version: Accepted Manuscript

Link(s) to article on publisher's website:  
<http://dx.doi.org/doi:10.1016/j.ijhcs.2014.09.001>

---

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online's data [policy](#) on reuse of materials please consult the policies page.

---

[oro.open.ac.uk](http://oro.open.ac.uk)

# Knowledge Transfer in Pair Programming: An In-depth Analysis

Laura Plonka, Helen Sharp, Janet van der Linden,

Centre for Research in Computing

The Open University Milton Keynes, UK

(Laura.Plonka; Helen.Sharp; Janet.VanderLinden@open.ac.uk)

Yvonne Dittrich

Software Development Group IT University of Copenhagen Copenhagen, Denmark (ydi@itu.dk)

## **Abstract**

Whilst knowledge transfer is one of the most widely-claimed benefits of pair programming, little is known about how knowledge transfer is achieved in this setting. This is particularly pertinent for novice–expert constellations, but knowledge transfer takes place to some degree in all constellations. We ask “what does it take to be a good “expert” and how can a “novice” best learn from a more experienced developer?”. An in-depth investigation of video and audio excerpts of professional pair programming sessions using Interaction Analysis reveals: six teaching strategies, ranging from “giving direct instructions” to “subtle hints”; and challenges and benefits for both partners. These strategies are instantiations of some but not all teaching methods promoted in cognitive apprenticeship; novice articulation, reflection and exploration are not seen in the data. The context of pair programming influences the strategies, challenges and benefits, in particular the roles of driver and navigator and agile prioritisation which considers business value rather than educational progression. Utilising these strategies more widely and recognizing the challenges and benefits for both partners will help developers to maximise the benefits from pairing sessions.

# 1 *Introduction*

1  
2  
3 “Two heads are better than one” is a common idiom referring to the advantages of collaborative work.

4  
5  
6 The value of collaboration is explicitly encouraged in software development through a practice known  
7  
8 as pair programming. Pair programming (PP) is a software development technique where two  
9  
10 developers work closely together to solve a development problem [41, 1].

11  
12  
13 Several benefits of PP have been claimed including improved understandability and maintainability of  
14  
15 code and design [35, 37], decreased defect rates [24, 18, 23, 10, 25] and knowledge transfer [21, 20, 23,  
16  
17 32, 34, 35, 36, 38, 40]. This paper focuses on knowledge transfer in PP. Indeed, Salinger et al [31] have  
18  
19 identified PP roles other than driver and navigator, including one called task expert which brings in task  
20  
21 expertise relevant to the session. This acknowledges the fact that all sessions include some level of  
22  
23 knowledge transfer. This acknowledges the fact that all sessions include some level of  
24  
25 knowledge transfer.

26  
27  
28 Most software development teams are composed of developers with different knowledge levels of some  
29  
30 kind, including different programming experience, different domain expertise and knowledge about  
31  
32 different technologies. PP is one way to share their knowledge with other team members while also  
33  
34 achieving meaningful work. In some cases, knowledge transfer is the explicit goal of a PP session [29].  
35  
36 This is common when a more experienced developer teaches a less experienced developer, for example,  
37  
38 to bring new staff up to speed [3, 42]. However, given that developers never have identical knowledge, a  
39  
40 certain degree of knowledge transfer would be expected within every PP constellation.  
41  
42

43  
44  
45 Pairing with someone who has a different knowledge level can be problematic [2, 7] and developers  
46  
47 tend to interact differently in this situation in comparison to pairing with other developers with similar  
48  
49 knowledge levels [9, 7]. For example, Plonka et al. [29] showed that less knowledgeable developers  
50  
51 (novices) can disengage in PP sessions and can sometimes not follow their more knowledgeable partner  
52  
53 (expert).  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

1 Although knowledge transfer is widely reported as a benefit of PP, there is currently not much insight  
2 into how developers approach this in practice nor how knowledge transfer can be improved. What does  
3 it take to be a good “expert” and how to learn best as a “novice”? What are the challenges? Here, we  
4 present an in-depth investigation of knowledge transfer in professional PP sessions to address the  
5 following research questions:  
6  
7  
8  
9  
10

- 11 • RQ1: What teaching strategies do developers use in pair programming?
- 12 • RQ2: In which ways do the roles of driver and navigator influence knowledge transfer in pair  
13 programming?  
14  
15  
16  
17  
18  
19  
20  
21 • RQ3: What challenges do developers with different knowledge levels face when pairing  
22 together?  
23  
24  
25

26 These three questions are addressed through a qualitative analysis (using Interaction Analysis [19]) of  
27 video recordings of professional developers working together on their day to day tasks. As a result, we  
28 identified a set of teaching strategies and behaviours that are related to the roles of driver and navigator  
29 and influence teaching and learning, together with associated challenges and benefits for both pairing  
30 partners. An increased awareness of working practices for knowledge transfer in PP will help developers  
31 to maximise the benefits from such sessions.  
32  
33  
34  
35  
36  
37  
38  
39

40 The remainder of the paper is organized as follows. Section 2 overviews existing research on knowledge  
41 transfer in PP. In section 3, we present the research methodology including data collection and analysis  
42 approach, followed by the findings of this study (section 4). In section 5, the findings are discussed with  
43 respect to existing literature and section 6 discusses the limitations of the study. The last section 7  
44 presents conclusions and implications for developers.  
45  
46  
47  
48  
49  
50  
51  
52

## 53 **2 Knowledge Transfer in Pair Programming**

54 The positive effect of PP on knowledge transfer, no matter what may be the knowledge levels of the  
55 developers, is widely acknowledged across a range of studies in industry [21, 20], [38, 35, 23] and  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

1 academia [32, 36, 34, 40]. Knowledge transfer is also one of the main perceived benefits according to  
2 two surveys: Schindler [33] surveyed developers and managers in 42 Austrian companies; and Begel  
3 and Nagappan [2] conducted a web-based survey of 487 Microsoft developers. Three industrial case  
4 studies [21, 35, 38] report more detail on developers' perceptions. In [21], developers report that PP  
5 increased their knowledge of the code and in [35], developers report increased knowledge of the  
6 software system. Gaining knowledge about development tools, work practices, refactoring old code, new  
7 technologies and programming languages are all perceived benefits reported in [38].

8  
9  
10  
11  
12  
13  
14  
15  
16  
17 Belshee [3] suggested very frequent changes of the pair constellation to promote fast knowledge transfer  
18 and to spread knowledge among different team members. Pandey et al. [23], suggests that this can  
19 reduce project risk because multiple developers are familiar with the code and there is less reliance on  
20 one individual. Increased flexibility also means that developers can pick up a variety of different tasks.  
21 For example, Hodgetts [17] reports on one team that had only one database expert, but too much work  
22 for one expert. When this caused a bottleneck, the team decided to use PP to spread the database  
23 knowledge among developers. They learned quickly through pairing with the database expert and were  
24 then able to do database tasks by themselves.

25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37 PP has also been studied in the context of training and mentoring, but not always with a positive effect.  
38 For example, in the context of developing firmware for processors, Greene [16] found that the training  
39 effect of PP was not as high as expected, which may be due to the very specialized and complex domain  
40 knowledge needed in that context. On the other hand, Williams et al. [42] investigated PP for mentoring  
41 and hence focused on pair constellations with different levels of expertise. They examined the  
42 relationship between PP and Brooks' Law<sup>1</sup> based on a survey and a case study. They found that PP  
43 reduced the mentoring needed per day from 37% of a developer's time to 26% of their time, and that PP  
44 reduced the time for a developer to be independently productive from 27 to 12 days.

---

45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59 <sup>1</sup> Brooks Law says that "adding manpower to a late software project makes it later" [6]  
60  
61  
62  
63  
64  
65

1 The developers' view of combining different knowledge levels when pairing was investigated by Jensen  
2 [18] and Vanhanen et al. [38, 37]. Jensen [18] found that pairing developers with similar expertise was  
3 counter-productive, while Vanhanen and Lassanius [37] found two good partner combinations: when the  
4 pair consists of a senior and a junior developer; or partners have complementary knowledge.  
5  
6

7  
8  
9  
10 When asked about the challenges of PP, developers surveyed by Begel and Nagappan [2] perceived  
11 working with someone with different skills as one of the main challenges. Williams and Kessler [41]  
12 also point out that pairing experts and novices can be problematic. Novices can slow down experts and  
13 some experts might not have a mentoring attitude.  
14  
15  
16  
17

18  
19  
20 One study by Cao and Xu [7] examined the interactions of pairs in more detail according to their  
21 expertise. They assigned students according to expertise and found that the expert asked for the novices'  
22 opinions frequently at the beginning of the session but stopped asking after realising that they did not get  
23 valuable information.  
24  
25  
26  
27

28  
29  
30 Although there is some evidence that pairing developers with different knowledge is useful but  
31 challenging, there is currently a lack of understanding about what interactions take place to achieve  
32 knowledge transfer and what challenges developers face.  
33  
34  
35  
36  
37

### 38 **3 Research Design**

39  
40  
41  
42 It is known that people working jointly on a computer use a combination of gesture, language and screen  
43 object manipulation to construct an understanding of the problem (see [30] for example). In PP  
44 developers work closely together on one computer and all these aspects needs to be considered when  
45 analysing knowledge transfer between the developers. For this study, we chose a data gathering  
46 approach that captures rich data about the PP sessions and an analysis approach that allows for a detailed  
47 investigation of how human beings interact with each other, and with objects in their environment (both  
48 verbally and non-verbally) [19].  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

### 3.1 Data gathering

Different aspects of the PP session were captured by using a combination of data gathering methods:

- Audio and video recordings were used to record the *developers' interactions* during their PP sessions: audio recordings of all verbal communication; a video of the programmers; and a full-resolution recording of the screen, showing the code and capturing the developers' computer activities. These were fully synchronized into a single video file (see figure 1).
- Questionnaires were used to gather *background information* about the developers, the aim of the session, and their experience in programming and PP.
- Interviews were conducted with both developers one day after the session to capture the *developers' account* of their session.

**Table 1:** Companies' and developers' background

Industry	Company size	Team size	Programming experience	PP experience
Geographic information systems	30-50	8	0.9-20 years	0-20 years
Traffic, logistic and transport	<500	2 teams, 5 developers each	0.4-13 years	0-3 years
Email marketing	50-100	8	1.3-10 years	0-5 years
Estate CRM Software	50-100	10	1.5-12 years	0-3.6 years

We recorded PP sessions from four different companies. All companies used agile approaches and all companies belonged to different industries. None of the companies provided developers with PP



1 training. Table 1 provides background about the companies and the developers. During our studies, the  
2 developers worked on their day-to-day tasks in their usual working environment. There was no set  
3 pattern for pairing. Developers decided themselves when it was appropriate to pair, and with whom. See  
4 figure 1 for an example of a recording set-up in one of the participating companies.  
5  
6  
7  
8  
9

### 10 3.2 Analysis

11 To analyze the data we used Interaction Analysis [19] which focuses on social interactions (verbal and  
12 non-verbal) as they take place in their natural settings through analysing everyday interactions. Video  
13 data supports Interaction Analysis because it captures the minutiae of interactions. Moreover, video data  
14 allows sequences of interactions to be replayed which is crucial for re-examining and understanding  
15 what happens in the session.  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25



40 **Figure 1:** Left: Screenshot of a fully synchronized video showing Eclipse IDE and the developers.  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

Jordan and Henderson [19] originally published their description of Interaction Analysis in the context  
of learning sciences. In recent years Interaction Analysis has been used in software engineering research.  
For example, Børte et al. [5] successfully used Interaction Analysis to study software effort estimation  
by investigating different types of knowledge, reasoning and decision-making in group based estimation  
sessions, while Dittrich and Giuffrida [15] used the method to investigate the role of instant messaging  
in a global software development project.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

Our analysis is based on the core procedures of Interaction Analysis but was tailored to the context of this study. The following summarises the six core procedures and how they were tailored, see [26] for details.

1. *Procedure: Ethnographic context* In order to provide context for the video data, Interaction Analysis suggests to capture the ethnographic context in which the recordings take place. For the data gathering of this study, the main researcher spent time at the organisations and in addition to the video data three other types of data (questionnaires, interviews, and field notes) were gathered during that stay.
2. *Procedure: Content logs* The intention of this procedure is to obtain an overview of the data through an initial viewing and annotation to create content logs. In this study, the video data was annotated through previous analyses [29, 27] and these annotations were used as content logs for this study.
3. *Procedure: Individual researcher's work* In this study, this procedure was divided into two separate steps. Firstly, the researcher extracted exemplar excerpts of the data (this process is referred to as “cannibalising” [19, p. 46]). An excerpt is regarded as an “exemplar” if it is a typical or representative example of the data being studied. Secondly, after the group work procedure (described below) the researcher reviewed the group viewing notes and analysed the interview data.
4. *Procedure: Transcription* The video data to be analysed is transcribed. This procedure was followed in this study and is described below in the second step of our analysis.
5. *Procedure: Group work* Group work is fundamental to Interaction Analysis. Group members discuss observations and hypotheses, searching for “distinguishing practices” and “identifiable regularities” in the interactions. In this study, this procedure was used in a slightly adapted form which is described in section 3.2.1.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

6. *Procedure: Video review sessions* In this procedure the video segments are played back to the participants. The intention of this step is to include the participants' perspective for the analysis to gather further insights. This procedure was not practical in the context of this study and was replaced by analysing the interviews that were conducted after the sessions and which provide the developers' perspectives on their PP sessions.

### 3.2.1 Analysis steps

In the following sections the specific analysis steps in the order in which they were conducted are described.

#### ***Step 1: Sampling of relevant video exemplars***

Interaction analysis is a very detailed and time-consuming analysis procedure which means that sampling of the relevant video exemplars is a crucial step. A detailed analysis of a 3-5 minute exemplar can take about 2 hours. At the start of this analysis, we had about 37 hours of video recordings of 21 PP sessions with 31 developers from four different companies. Each session lasted between one and a half and three and a half hours. In previous studies [29, 27] we had analysed the full 37 hours using different analysis methods. Hence, we had excellent knowledge of the data before the Interaction Analysis started. This allowed us to effectively identify exemplars that exhibit typical situations in which knowledge transfer between developers takes place. Five suitable exemplars were identified, each between 4 and 6 minutes in length, that together represented the set of typical situations in the data. The following criteria guided our selection.

- *Developers with different levels of knowledge are aiming to transfer knowledge.* Developers never have exactly the same knowledge, and so it can be assumed that knowledge transfer takes place in every PP session. However, in PP sessions where developers have similar levels of expertise, knowledge transfer and the strategies used to achieve it might be difficult to identify.

In contrast, where pair constellations have an explicit aim to transfer knowledge from one

1 developer to another, knowledge transfer and associated strategies can be observed more easily  
2 and explicitly. Moreover, in these sessions it is clear who is teaching whom. Hence, we used the  
3  
4 background questionnaires and the interviews with the developers to identify sessions in which a  
5  
6 more knowledgeable developer (expert) worked with a less knowledgeable developer (novice).  
7  
8

9  
10 In the context of this study, the definition of an expert and a novice is based on the developers'  
11  
12 perceptions of their knowledge for a particular pair programming session. We chose excerpts  
13  
14 from PP sessions for which developers explicitly stated that the aim of the session was to transfer  
15  
16 knowledge from the more knowledgeable developer (expert) to the less knowledgeable  
17  
18 developer (novice) for the topic covered in this session. In addition we only chose sessions in  
19  
20 which both developers agreed who is the expert and who is the novice in this particular session.  
21  
22

- 23  
24  
25 • *Experts are trying to teach novices.* To address RQ1 excerpts were selected in which experts  
26  
27 tried to teach the novices rather than excerpts in which no communication, explanations or  
28  
29 verbalisations took place.  
30  
31
- 32  
33 • *Both expert and novice are driving within the excerpt.* To answer RQ2 some excerpts were  
34  
35 chosen in which the novice was driving and some in which the expert was driving or where the  
36  
37 developers switched roles.  
38  
39
- 40  
41 • *Behaviour is not unique to one pair constellation.* Although we only selected a small number of  
42  
43 exemplars to conduct the in-depth interaction analysis the first author watched all video  
44  
45 recordings to ensure that the selected excerpts represent behaviours observed in multiple pair  
46  
47 constellations.  
48  
49

## 50 51 **Step 2: Transcription**

52  
53  
54 The conversations of the developers together with timestamps and pauses were transcribed. Short pauses  
55  
56 in the communication are marked as [...] and for longer pauses the number of seconds [sec] is provided  
57  
58 in the transcriptions. The collaborator companies were based in Germany and all audio data was in  
59  
60  
61  
62  
63  
64  
65

1 German. The transcription was based on the original data. The main researcher who conducted the data  
2 gathering is a native German speaker and the other researchers have different language skills (see table  
3  
4  
5 2).  
6  
7

### 8 ***Step 3: Group viewing*** 9

10 The video excerpts were analysed collaboratively during group viewing sessions. The group size varied  
11 between two and five members. In each group viewing session at least one German and one non-  
12 German speaker were present. Table 2 provides an overview of the members of the group, their  
13 language skills and their relevant experience. Each group member had a different background. This  
14 strengthened the analysis as each member provided a different perspective on the data. For example, the  
15 different language skills of the group members influenced the initial focus when watching the excerpts;  
16 usually, the non-German speakers focused first on the non-verbal communication and on the computer  
17 activities of the developers while the German-speaking members focused on the conversation first. To  
18 allow every member to understand the full picture of the events the transcripts were translated during the  
19 group viewing session.  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35

36 For each video excerpt (4-6 min) the group session took approximately 2 hours in order to accommodate  
37 intense discussions. During the whole group viewing session the first author took detailed notes of the  
38 discussion. The group viewing session had the following activities (steps 3 and 4 were iterated):  
39  
40  
41  
42

- 43 1. *Watching the whole video excerpt* The group watched the excerpt from the beginning to the end  
44 without stopping. The group was not provided with the context of the excerpt before watching it,  
45 in order to counter researcher bias.  
46  
47  
48
- 49 2. *Discussing initial observations* Each team member shared their initial observations with the  
50 group, drawing on their specific expertise.  
51  
52  
53  
54  
55
- 56 3. *Providing context for the segment* After the initial discussion, the group members were told the  
57 context of the excerpt: who is the expert and who is the novice; is this excerpt from a beginning,  
58  
59  
60  
61  
62  
63  
64  
65

middle or end of a session; and what developers reported in the interviews about their knowledge transfer experience.

4. *Stop-and-go watching of the video excerpt* The excerpt was watched again but this time, team members stopped the video to discuss different dimensions of the interaction in detail. During this, certain events or the whole excerpt were re-played several times.
5. *Finding themes* The results of group viewing sessions were brought together and the emerging themes were discussed.

**Table 2:** Members of the interaction analysis group

Research experience	Relevant languages	Relevant experience
Experienced researcher	English	Empirical software engineering, qualitative research, research on agile methods
Experienced researcher	German (intermediate), English	Empirical research, qualitative research, gesture analysis, advanced programming
Experienced researcher	German, English	Empirical research on cooperative and human aspects of software engineering, interaction analysis
PhD student	English	Empirical research, qualitative research
PhD student	German, English	Empirical research, qualitative research, pair programming research

The analysis was exploratory and did not follow a pre-defined coding scheme. During the group viewing, observations were not restricted to the research objectives because a restriction early on in the

1 analysis process might lead to important aspects being overlooked. Once the group decided that all  
2 relevant observations were discussed, the group viewing session was over and the next group session  
3 focused on a new excerpt.  
4  
5  
6  
7

#### 8 ***Step 4: Analysis of group viewing notes and interview data*** 9

10 The themes that emerged during the group viewing were reviewed with respect to the research  
11 questions. The first author reviewed the extensive notes from the group session and revisited the data for  
12 each theme for validation. The results of this procedure were discussed with other group members. In  
13 addition, the interviews from the three selected PP sessions were analysed focusing on developers'  
14 statements related to their experience of knowledge transfer within their sessions. The findings from the  
15 interviews were then used to contextualise the findings from the group viewing session.  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25

## 26 **4 Findings** 27 28 29

30 This section presents the findings focusing on the teaching strategies used by developers (RQ1) the roles  
31 of driver and navigator (RQ2) and the challenges when pairing developers with different knowledge  
32 levels (RQ3). Our findings are illustrated through examples from the selected transcriptions and  
33 descriptions of what developers were doing. The analysis highlighted that experts use a combination of  
34 different strategies to teach novices and that each expert uses a variety of strategies even within the same  
35 pair programming session.  
36  
37  
38  
39  
40  
41  
42  
43  
44

### 45 **4.1 What teaching strategies do developers use when a novice is driving?** 46 47 48

49 We identified four different strategies that experts use when the novice is driving; Verbal nudging and  
50 physical hints, pointing out problems, gradually adding information, and giving clear instructions.  
51  
52  
53

#### 54 **4.1.1 Verbal nudging and physical hints** 55 56

57 Verbal nudging and physical hints are teaching strategies that provide directions without providing the  
58 solutions for the novice. For example, using verbal nudging an expert will make suggestions rather than  
59  
60  
61  
62  
63  
64  
65

1 explicitly tell the novice what to do. Other subtle guidance can include physical hints such as pointing to  
2 something on the screen or placing the cursor in a particular location. These different types of  
3 behaviours are illustrated in Example 1 and Example 2.

4  
5  
6  
7 In Example 1, the novice is driving and has just finished writing a line of code that checks an entry in a  
8 list. The novice suggests that they can now move on (line 1) but the expert proposes (using the word  
9 “could” in line 2) that they should test this code first. The novice agrees without any resistance and  
10 instead of moving on, the novice starts testing the code.  
11  
12  
13  
14  
15

### 16 17 18 **Example 1 Nudging**

19 20 21 <b>Line</b>	<b>Speaker</b>	<b>Talk</b>
22 23 24 25 26 27 1	Novice:	“Ok, so this is done now, so we can move on to the next bit.”
28 29 30 31 32 33 34 2	Expert:	“We could also test that first.”
35 36 37 38 39 40 41 42 43 44 3	Novice:	“Yes, ok.”

45  
46  
47 In Example 2, the expert places the cursor in a certain position before handing the keyboard over to the  
48 novice. Later in the excerpt, it becomes apparent that this is the point in the code where the problem  
49 should be addressed. The fact that the expert switched to this test class after finishing his explanations  
50 and while preparing the handover, indicates his intention to provide a hint for the novice.  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

### 66 67 68 **Example 2 Indirect hint: Preparing the environment**

69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000
--



1 The excerpts illustrate that verbal nudging and physical hints are used by experts to provide a learning  
 2 opportunity for the novice. In Example 1, the expert uses a gentle form of verbal nudging which is  
 3  
 4 immediately picked up by the novice. Without telling the novice how exactly to solve the problem, the  
 5  
 6 expert provides successful directions. Following the excerpt in Example 1, the novice starts writing a  
 7  
 8 test. In addition to verbal hints, we also observed indirect non-verbal hints where an expert physically  
 9  
 10 moves the cursor around the programming environment to nudge the novice towards the right place.  
 11  
 12 Example 2 illustrates such a situation in which the expert sets up the environment for the novice before  
 13  
 14 handing over the keyboard to provide the novice with a starting point. These subtle strategies were  
 15  
 16 observed in pairs where the expert seemed to be patient and the novice had some initial knowledge of  
 17  
 18 the task at hand.  
 19  
 20  
 21  
 22  
 23  
 24  
 25  
 26

#### 27 **4.1.2 Pointing out problems**

28  
 29 Experts point out problems for novices without suggesting how the problem should be solved. This is  
 30  
 31 illustrated in Example 3.  
 32  
 33  
 34  
 35

36 **Example 3** Pointing out a problem

37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65	Line	Speaker	Talk
	The novice has just finished writing some code.		
	1	Expert:	"I see at least three mistakes."
	2	Novice:	"You see three mistakes?"
	3	Expert:	"I see three mistakes."
	4	Novice:	"OK."
	5	Expert:	"Twice <code>statement.close</code> and one uninitialized member variable."
	6	Novice:	"Yes."

This is followed by the novice suggesting how to address the problems.

In Example 3, the novice has finished writing some code. The expert points out that there are at least three mistakes in it without explicitly explaining how to address them. The expert thus gives the novice the space to think about how to solve the problem without simply solving it for him and the novice starts suggesting how to address the problems. In comparison to verbal nudging and physical hints, in the example above the novice receives no direction on what to do next. This strategy prompts the novice to suggest how to address the problems, giving the novice the opportunity to think the problems through by herself/himself. We observed that this strategy can be time-consuming because it might take the novice some time to identify solutions to the problem. In cases where this strategy does not work, we observed that experts use a follow-up strategy (gradually adding information). This strategy is presented next and is also used independent of the pointing out problems strategy.

#### 4.1.3 Gradually adding information

Gradually adding information means that the expert supports the novice in finding a solution for a problem on an “as needed” basis. Instead of suggesting how to solve an issue experts wait and see whether novices are capable of solving a problem by themselves with a certain amount of information given. If the novice is not capable of solving the task, the expert gradually adds more information in order to help the novice (as illustrated in Examples 4 and 5).

##### Example 4 Gradually adding information (1) and Giving clear instructions

Line	Speaker	Talk
1	Novice:	“We would maybe need a constructor here, wouldn’t we?”
2	Expert:	“Right. That would be good.”

3	No communication, no typing [3sec].	
4	Expert:	“At the top.”
Novice is moving the cursor to the top of the class.		
5	Expert:	“ALT-N.”
Novice presses ALT-N.		

**Example 5** Gradually adding information (2)

Line	Speaker	Talk
1	Novice:	“Ok, technically, we’d have to call a [...] job instead of calling a PreSQLStatements.”
No communication/no driving [3sec].		
2	Expert:	“In our case, we’d call a [...] job instead now. Right.”
3	Novice:	“Right. Ok, that means we somehow build us a class now that then just gets the job and executes it.”
4	Expert:	“Eventually, that would be the implementation. Yes.”
5	Novice:	“Good.”
No communication/no driving [2sec]		
6	Expert:	“ We just have to tell the SQL FilterStatement somehow [...] that it executes it at the right time.”
No communication/no driving [3sec]		
7	Expert:	“Instead of using PreSQLStatement; [...] use PreSQL or PreListener or PostListener or all in one. I don’t know.”
8	Novice:	“Yes. Ok, but we would want to do that then by using the Config-class.”
Novice puts his hands on the keyboard and starts typing.		

9	Expert:	“Right. That is our only chance to intervene there.”
---	---------	--

1  
2  
3 In Example 4, the novice is driving and suggests creating a constructor. He phrases this suggestion as a  
4 question (line 1). The expert confirms that the novice’s suggestion is right without adding more  
5 information (line 2). The novice hesitates to start typing. After a short pause, the expert gives the novice  
6  
7 additional instructions by telling him that the constructor should be written at the top of the class and  
8 that he should use the shortcut “ALT-N” to do that.  
9

10  
11  
12 This can be interpreted as the expert waiting to see whether the novice can create the constructor without  
13 more information and only giving instructions when the novice hesitates.  
14

15  
16  
17 Example 5 presents a second example of this strategy. Just before the start of this excerpt, the expert had  
18 explained the code and the problem and then made it verbally explicit that it’s the novice’s turn to drive.  
19

20  
21  
22 The novice does not immediately take the keyboard or mouse. Instead the novice starts suggesting what  
23 to do next (line 1 and 3). In lines 2 and 4, the expert repeats and confirms the novice’s suggestion  
24 without adding new information. When it is the novice’s turn to talk and type, the novice hesitates again.  
25

26  
27  
28 The expert starts adding further information about how to solve the problem. After adding new  
29 information (line 6) the expert waits. The novice does not react and after another moment of silence the  
30 expert starts again to add new information (line 7). This time (line 7) the expert finishes his statement by  
31 saying that he does not know what the best approach would be. In line 8, the novice suggests how to  
32 address the problem and subsequently takes the keyboard and starts implementing the solution. The  
33 expert confirms the novice’s suggestion in line 9.  
34

35  
36  
37 Later on in the session (not provided as a transcript), it becomes clear that the expert knew how to  
38 approach that problem, and that the “I don’t know” statement in line 7 was a hint that the novice should  
39 solve the problem.  
40

41  
42  
43 Both examples 4 and 5 show an expert not giving all the information to the novice at once but gradually  
44 adding more information when necessary. This suggests that the expert wants the novice to actively  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

1 think about the problem rather than for him to just explain each step immediately. This also means that  
2 the problem solving process might take more time in comparison to having the expert telling the novice  
3 explicitly how to solve the problem (as described in the next section).  
4  
5  
6

#### 7 8 **4.1.4 Giving clear instructions** 9

10 Experts also use clear and direct instructions, both with and without additional explanations, to help  
11 guide the novices. These instructions include dictating what to type, which shortcuts to use and where  
12 the changes have to be made in the code. In some cases, experts explain their instructions thereby  
13 providing the novices with the reasoning behind those instructions. In both cases, it was observed that  
14 the novice follows the instructions of the expert immediately.  
15  
16  
17  
18  
19  
20  
21  
22

23 Looking back at example 4, this acts as a good example of an expert giving clear instructions when  
24 noticing that the novice knows what to do (line 1), but hesitates about how to go about it (line 3). The  
25 expert gives the novice clear instruction where to create it (line 4) and how to do it (“ALT-N” in line 5)  
26 without providing any additional explanation.  
27  
28  
29  
30  
31  
32

33 A similar example of providing clear instructions on what to do is shown in example 6. However, here  
34 we also note the expert providing an explanation on why the code had to be placed elsewhere (line 2).  
35  
36  
37  
38

39 In contrast to the three previous strategies, in this strategy the experts solves the problem for the novice.  
40 This strategy doesn’t encourage the novice to solve the problem for himself/herself. This might be less  
41 time-consuming but also provides the novice with less opportunity to explore different approaches.  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51

#### 52 **Example 6 Giving direct instructions with explanations** 53

54 1	55 Novice:	56 “Yes, so then ... we can create a nice method now.”
57 Novice start to create a new method. 58 59		

2	Expert:	“You better do that above because this here is an inner class.”
Novice deletes the already written code and creates the new method outside of the inner class.		

## 4.2 What teaching strategies do developers use when expert is driving?

Focusing now on strategies where it is the expert who is driving, rather than the novice as in the previous section, we identified two strategies: explanations and verbalisations. It is difficult to differentiate between explaining to oneself (verbalisation) and explaining interactively [14]. In this paper, verbalisation refers to verbalising while performing activities without being asked for it, while explanations are statements triggered by a question or a comment that makes it clear that the other person needs additional information.

### 4.2.1 Explanations

When the novice asks for an explanation, experts can address the question verbally (Example 7) or by showing the novice on the computer how to do certain steps (Example 8).

#### Example 7 Explanation

Line	Speaker	Talk
1	Expert:	“So, and now we create some methods and call them test. <code>BasicFilter</code> , isn't it?”
2	Novice:	“Was the style back then so different that, hm [...], that err... all tests were in one method?”
Expert keeps typing and replies while typing.		
3	Expert:	“Err.. no in terms of style not but err that happened quite often back then just because err...”
Expert keeps typing for 5 sec without any comments and then briefly verbalises what he is typing		
4	Expert:	“ <code>PreparedStatement</code> with S, erm

Expert stops verbalizing and keeps typing for the next 10 sec without saying anything		
5	Expert:	“Because one was lazy and just wrote it like this and the strict policy did not exist either.”
4	Novice:	“OK.”
6	Expert:	“The number of parameters for the methods did not exist like today either.”
Expert keeps writing for the next 9 sec without saying anything		
7	Expert:	“Oh wow, they [methods] do all build upon each other.”
Expert keeps writing for the next 9 sec without saying anything		
8	Expert:	“ Ah that is stupid, that is no fun.”

**Example 8** Explanation by showing

Line	Speaker	Talk
1	Expert:	“So, now you can go over it and see whether it finds it. Have a look whether the activities exist. So, you can [...]”
Novice leans a little bit back, takes her hand from the mouse, turns around, looks at the expert and says:		
2	Novice:	“The activities?”
Expert leans forward, takes the mouse and shows the novice how to use the debugger to check the values for the list entries. He removes his hand from the mouse, leans back again and puts his hand under the table.		

Example 7 illustrates how the expert addresses the novice questions verbally. In the excerpt, the developers are amending tests that have been written before. The expert is driving and the novice notes that the structure of the old tests does not conform to the current coding policy. He asks (line 2) the

1 expert about the different styles. The expert starts explaining how the coding standards evolved and at  
2 the same time expresses his displeasure about the old structure of the existing code.  
3  
4

5 In Example 8 the expert initiates a short role switch to show something to the novice. The novice is  
6 driving and the expert suggests what the novice should do next, which requires knowledge about the  
7 debugger. The novice does not seem to know how to do that. The novice leans back, looks at the expert  
8 and repeats the keyword “activities” as a question (line 2). The expert takes the mouse, shows the novice  
9 what he meant and takes his hands immediately back from the mouse.  
10  
11  
12  
13  
14  
15  
16

17 In both cases, the expert addresses the novice’s request for information. Depending on the information  
18 request it seemed to be easier to explain the knowledge verbally (for example, how do certain processes  
19 work, why have specific decisions been made) while in other cases it seemed to be easier, more  
20 convenient and maybe less time consuming to show steps rather than explain them.  
21  
22  
23  
24  
25  
26  
27

#### 28 **4.2.2 Verbalisation**

29  
30

31 Experts verbalise their activities and thoughts while driving. Verbalisation is not necessarily directed to  
32 the novice, but might help the expert to structure his/her own thoughts; as a result, it may help the  
33 novice to understand the expert’s thought process.  
34  
35  
36  
37  
38

#### 39 **Example 9 Verbalisation**

40  
41

42 <b>Line</b>	43 <b>Speaker</b>	44 <b>Talk</b>
45 1	46 Expert:	47 “Now, let’s have a look. I just open the tests. This is where I had started 48 already. I just wanted to show that to you and then you may (drive) as 49 well.”
50 Expert opens the test and then starts browsing through the code.		
51 2	52 Novice:	53 “And this (test) always has everything in it? It will get big.”
54 3	55 Expert:	56 “Right, yes, that will get a bit bigger. I leave it all in here for now and 57 then we can think about whether we should create a second test class.” 58 59

60  
61  
62  
63  
64  
65



4	Novice:	“And for this test, do all the other tests have to be run through before? Or could we get a specific one?”
5	Expert:	“So, I thought, we have this “before” here. Let’s go through it again. It creates one <code>CommissionCalculation</code> . That is here and it ... the DNS. This [experts navigates with mouse through the code] is adding a hint, this is adding the “invoices”, and this one is adding a <code>First-WorkFlow</code> among others. It sets the status in <code>created</code> , <code>added</code> and sets this as <code>current Workflow</code> .”
6	Novice:	“Yes, this is what it just did.”
7	Expert:	“And we also made sure it synchronises this automatically. Add the “recipients”, err [...] maybe that is not as important at the moment.”

In Example 9, the expert had worked a particular part of the code before and now goes through it again in order to explain the code to the novice (line 1). After some browsing through the code, the novice asks a question about the size of the test (line 2) and while the expert responds (line 3) he indicates that they should not focus on the size issue. During this exchange the expert is very focused on the screen (his eyes never leave the screen) and on understanding the code that he had written before. The subsequent question by the novice (line 4) is in fact ignored by the expert (line 5) who proceeds to verbalize his thought process about how the code works.

The fact that he does not react to her comments indicates that he verbalises his thoughts for himself rather than to provide explanations for her, because if his main interest was to explain the code to her, he would pay attention to her comments. However, he stated in line 1 that he does want to explain the code to her, so it may be that he is concentrating on verbalising his thoughts and can’t take her viewpoint into account. This stresses that there might be a conflict between self-verbalisation and verbalising for the partner (this is discussed in detail in section 4.4).

### 4.3 In which ways do the roles of driver and navigator influence knowledge transfer in pair programming?

The role being taken had a clear influence on the developers' interactions, and on the novice's engagement. In a previous study [29], we have shown that novices are at risk of disengaging from the PP session when navigating and when there is a lack of communication. In contrast here we focus on excerpts where experts and novice communicate and on excerpts when the novice is driving. We observed that the novices tend to be more active when driving. This shift in behaviour is illustrated in the next section. It also became apparent that novices are articulating what they doing when driving to get reassurance from the expert as described in section 4.3.2.

#### 4.3.1 From a listener to an active participant

We observed that novices act more like listeners when navigating and turn into active participants that verbalise their behaviours and ask questions when driving. Examples 10 and 11 describe role switches, demonstrating the contrast between the novice's behaviour when navigating with that of being the driver. Example 11 is the same as Example 5, but it is repeated below to emphasise that it is a continuation of Example 10 in the PP session.

#### Example 10 Role switch

Line	Speaker	Talk
Line 1-10: Expert is driving. He keeps moving the cursor to the code fragments that he explains. The novice has his hands under the table.		
1	Expert:	"And that of course, leads to the point that not all entries are used, but [...]"
2	Novice:	"Mhm, ok."
3	Expert:	"Or if you call <code>execute</code> twice at the same <code>Prepared-Statement</code> [...]"
4	Novice:	"OK."

5	Expert:	“And that of course leads to the fact that even when the fields are not used [...]”
6	Novice:	“OK.”
7	Expert:	“So and we do the same somehow when the method is <code>close</code> and we [...]”
8	Novice:	“Mhm.”
9	Expert:	“That is the code that we have [...]. What we want to do now - aside from getting the big learning effect - err, is of course, eh, to instead of just using SQL, to ehm, it would be easier [...]”
10	Novice:	“Mhm.”
11	Expert:	“OK? So then we can change driver now..”
Expert takes the hand from the mouse, leans back and grabs a drink. The novice puts his hands on the table but not on the keyboard or mouse.		

**Example 11** Role switch (continued)

Line	Speaker	Talk
12	Novice:	“Ok, technically, we’d have to call a [...] job instead of calling a <code>PreSQLStatement</code> .”
No communication/no driving [3sec].		
13	Expert:	“In our case, we’d call a [...] job instead now. Right.”
14	Novice:	“Right. Ok, that means we somehow build us a class now that then just gets the job and executes it.”
15	Expert:	“Eventually, that would be the implementation. Yes.”
16	Novice:	“Good.”
No communication/no driving [2sec]		
17	Expert:	“We just have to tell the <code>SQLFilterStatement</code> somehow [...] that it

		executes it at the right time.”
No communication/no driving [3sec]		
18	Expert:	“By instead of doing <code>PreSQLStatement [...]</code> , <code>PreSQL</code> or <code>PreListener</code> or <code>PostListener</code> or all in one. I don’t know.”
19	Novice:	“Yes. Ok, but we would want to do that then by using the <code>Config-class</code> .”
Novice puts his hands on the keyboard and starts typing.		
20	Expert:	“Right. That is our only chance to intervene there.”

In the first 11 lines, the expert is driving and explains the code to the novice, using the mouse to highlight code snippets that are relevant for his explanation. While the expert is in control of the mouse, the novice keeps his hands under the table indicating that he is not trying to take on the role of driver. The novice appears to be an active listener but he is not asking any questions. In line 11, the expert mentions that it is time to switch roles. This is when the body language and the verbal involvement of the novice changes. The novice puts his hands on the desk but does not take the mouse or keyboard. He starts asking questions and makes suggestions about the actual implementation (Example 11 lines 12, 14, 19). The expert initially repeats his suggestions without adding new information and then adds information gradually (section 4.1.3). Eventually, the novice takes the keyboard and starts typing.

This example illustrates the novice’s change of behaviour from being a passive participant (as navigator) to a more active participant (as driver). As a navigator it might be enough to understand the underlying concepts but as a driver detailed knowledge is required of the code and the next steps are required to perform the implementation. This means that letting the novice drive can be useful to ensure that detailed knowledge is transferred. However, it also means that the process of solving the task at hand might be slower as the novice will need detailed information about the task and possible solutions.

### 4.3.2 Articulation and re-assurance

Novices tend to articulate their thoughts and verbalise their steps, their suggestions for solutions and reasoning behind them. A closer look at the verbalisation shows that novices seek for re-assurance about their actions from the experts. This is illustrated in Example 12. The novice verbalises her plans for the next steps (lines 1 and 3) and also asks for reassurance (line 5). The comments and questions are very detailed. This shows that letting the novice drive can encourage both expert and novice to be actively involved in discussing, understanding and solving the task at hand.

#### Example 12 Novice driving

Line	Speaker	Talk
Developers just switched roles and the novice takes the keyboard and starts creating a for-loop to check entries in a list.		
1	Novice:	“So, now here we have a “subject”. [...] “From”. [...] So, I just go through [the list] now and when I find one [activity], what am I doing with it then?”
2	Expert:	“Technically, it should only find one, if we were good, then it should have only one “activity” with that “subject”.
Novice deletes the for-loop.		
3	Novice:	“Then, then I say “assert”.”
4	Expert:	“Right.”
5	Novice:	“So we just check afterwards? ”
6	Expert:	“Technically yes. Just, technically, it would be enough that there is exactly one “activity”.”
7	Novice:	“Ah, ok, in the test I can see, that, what is in there. I’m still with my old <code>Sys.Out</code> .”
8	Expert:	“No, we can debug that.”

## 4.4 What challenges do developers with different knowledge levels face when pairing together?

In a previous study [29], we analysed and presented challenges faced by novices in expert–novice constellations. For this paper, we focus on the challenges that experts face when trying to transfer knowledge. Experts seem to face challenges when they are driving and guiding the novice at the same time.

### 4.4.1 Conflict between self-verbalisation and communication with partner

Example 9 illustrates the conflict of self-verbalisation and communication with partner. It is particularly interesting because the expert was intending to explain the code to the novice (line 1) but ends up focusing on understanding the code himself.

As the session continues (shown in Example 13) the expert still focuses on the code and on his approach to solving the problem. The novice has a different problem solving approach and tries to communicate that idea to the expert but is not immediately successful. The expert reacts to the novice’s comments but does not really take the novice’s suggestions into account. However, then the expert realises that the novice had the right idea (Example 13, lines 7 and 10).

#### Example 13 Verbalisation (continued)

Line	Speaker	Talk
1	Expert:	“The first checkpoint should have gotten an email already. Just because I saved it here in the status <code>edit</code> [...] because it is now in status <code>created</code> ”
2	Novice:	“None.”
3	Expert:	“ <code>Created</code> and <code>None</code> . ”
4	Novice:	“Yes, <code>Created</code> and <code>None</code> . ”

5	Expert:	“That means, if our logic is correct, what we just programmed, then we could indeed already search for an “activity”, that has the subject “from”, err, err from a work-flow that goes to the first checkpoint.”
6	Novice:	“Ah, but I understood that at the interface, the creator, so that the calculation will be delivered to the creator. Hence, the interface, the “wizard” sets the “workflow” to “first prove”. And we are at “first prove”. That means it must have passed it on.
7	Expert:	“You’re right. I think this is how we did it here, didn’t we? I have to have a look again.”
Expert navigates to another window.		
8	Expert:	“So, at the moment, here I can orientate myself better.”
9	Novice:	“Yes, yes.”
10	Expert:	“You’re right. For sure. That means that nothing has happened here yet. You’re right, completely right.”

The behaviour of the expert indicates that he is not able to cope simultaneously with the novice’s suggestions, with understanding the code and structuring his thoughts. However, neither of the developers seems to realise that; the expert does not ask for time to finish his thoughts nor does the novice stop making suggestions.

This issue of thinking and communicating at the same time was also identified by experts during the interviews: *“...complex, analytical and problems related to architecture. I prefer to work those kind of problems through in my head first. [...] My point is I cannot communicate and share my thoughts when I have to think about complicated problems.”*

#### 4.4.2 The effort needed to explain

Providing explanations for the novice can be an additional effort for the expert.

1 In Example 7 the expert explains former coding conventions while working on the current tests. Hence,  
2 the expert has to switch context between his current activity of working on tests and his explanation  
3 about former coding conventions. In this case, the explanation is not directly related to his current  
4 activity. The expert's explanations about the former coding conventions are frequently interrupted by  
5 breaks (err or pauses) and by verbalising what he is currently typing (line 4). This indicates the effort of  
6 switching the context between his task and his explanation. Furthermore, his typing slows down while  
7 he is explaining.  
8  
9

10  
11  
12 In contrast, Example 8 provides an episode in which the explanation appears to be effortless for the  
13 expert. In that situation, both developers focus on the same problem and the expert explains the use of  
14 the debugger which is directly related to his current activity. This indicates that explanations that are not  
15 related to the current activity can lead to an additional cognitive effort.  
16  
17

#### 18 **4.4.3 Benefits of verbalisation and explanation**

19  
20 Analysis of the interviews showed that novices can help experts to challenge their assumptions and  
21 reflect on the existing code and that working with a novice provides them with opportunities to learn  
22 themselves. These quotes from two of the interviews illustrate the developers' perspectives:  
23  
24

25  
26  
27  
28 *"It is good to work with her. She is always asking the right questions. I don't perceive that as slowing*  
29 *me down [...] so she is asking automatically the right questions and that forces me to think about what*  
30 *I'm actually doing."* and  
31

32  
33  
34  
35  
36  
37  
38  
39 *"It is really helpful to work with him. When I work with someone who is already familiar with the code,*  
40 *the risk is that we overlook things because we have always done it like this [...] That happens less when*  
41 *working with a newbie because he does not know these parts of the code and so he asks questions about*  
42 *it. And then I feel like I'm being forced to reflect and to explain what the software is doing."*  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65



## 5 Discussion

This section returns to the three research questions.

### 5.1 RQ1: What teaching strategies do developers use in pair programming?

Six strategies which experts combine to guide and teach novices in PP sessions emerged from our data analysis: (1) indirect hints, (2) pointing out problems, (3) gradually adding information, (4) giving clear instructions, (5) explanation, and (6) verbalisation.

The knowledge transfer aspect of pair programming can be viewed as a kind of apprenticeship, but traditional apprenticeship involves learning a physical activity [11, 13] through social interactions while focusing on a task. Cognitive apprenticeship, on the other hand, focuses on learning “cognitive and meta-cognitive, rather than physical skills and processes” [12, p. 3]. Collins et al. [12, 11] stress the importance of making tacit processes visible for learners by making thinking visible. Given that software development requires cognitive and meta-cognitive skills and that the key to understanding software development is the reasoning and concepts behind it, rather than the physical act of typing, cognitive apprenticeship shares the same characteristics, and is therefore relevant to learning and improving software development skills.

Comparing our strategies with the teaching methods suggested in cognitive apprenticeship [12], they can be viewed as specialisations of the teaching methods described there. This then provides further insights into strategies that might be used in PP as described below where we systematically relate the knowledge transfer strategies that emerged from our data with the teaching methods put forward by

Collins:

Comparing our strategies with the teaching methods suggested in cognitive apprenticeship [12], and viewing them as specialisations of the teaching methods described there provides insight into strategies that might be used in PP. The six teaching methods suggested by Collins et al. and how they relate to the six knowledge transfer strategies are described below.

1 **Modeling** refers to the expert demonstrating a task and verbalising his or her thoughts at the same time  
2 to make the process of thinking visible. Strategies (5) and (6) are examples of Modeling: experts  
3 verbalised their thoughts while the expert is driving, and explained more when asked to.  
4  
5

6  
7 **Coaching and Scaffolding**<sup>2</sup> describes the process of observing the learner while solving a task and  
8 providing support. Strategies (1)-(4) are examples of Coaching and Scaffolding used when the novice  
9 was driving. Strategies (1) and (2) make the novice solve the problem themselves. Strategy (3) helps to  
10 identify how much help the novice needs. Scaffolding should consider the current skill level of the  
11 learner as described by Vygotsky's [39] "Zone of Proximal Development". He says that learners might  
12 be able to solve a task in collaboration with a more capable peer that they would not be able to solve  
13 independently. In contrast, strategy (4) does not enforce the same level of novice engagement. "Giving  
14 clear instruction" is a less time-consuming Coaching approach but learning may suffer as the novice is  
15 not thinking for themselves.  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28

29 **Articulation** refers to learners being encouraged to articulate their knowledge, reasoning and problem-  
30 solving processes as Articulation refines the learner's understanding. In none of the sessions did the  
31 expert explicitly encourage the novice to articulate their knowledge. Some form of Articulation was  
32 observed when the novice was driving (4.3.2). In this case, it seemed that articulation was used as a  
33 method to get reassurance from the expert rather than to make thinking visible.  
34  
35  
36  
37  
38  
39  
40  
41

42 **Reflection** means that learners compare their own problem-solving process with those of others. This  
43 behaviour was not observed during our study.  
44  
45  
46

47 The method "**exploration**" is not included in the discussion because this method refers to helping a  
48 novice to choose suitable follow-up tasks to foster and advance their learning, and identifying such tasks  
49 would not normally be decided during PP sessions.  
50  
51  
52  
53

---

54  
55  
56 <sup>2</sup> The strategies Coaching and Scaffolding were merged because Scaffolding is one form of Coaching  
57 and no clear delineation is provided  
58  
59  
60  
61  
62  
63  
64  
65

1 So without any prior training, experts in a PP session are using cognitive apprenticeship teaching  
2 methods for knowledge transfer. But why don't pair programmers use all of these strategies? Firstly,  
3 developers may just not be aware of cognitive apprenticeship. Secondly, Collins describes the methods  
4 in the context of education which differs from the context of PP. Although developers work together  
5 with the explicit aim to transfer knowledge from an expert to a novice, tasks are typically chosen  
6 according to agile prioritisation which considers business value, not educational progression. Moreover,  
7 knowledge transfer is not the exclusive aim of the session because developers work on their real-world  
8 tasks with the focus of finishing the project on time. This means that developers have to balance  
9 knowledge transfer and getting the task done effectively.  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21

22 One strategy from cognitive apprenticeship that could be applied to improve knowledge transfer in PP is  
23 that of articulation. Encouraging the novice to articulate was not observed in our PP sessions, yet it  
24 could expose novice's thoughts. Vygotsky's [39] "Zone of Proximal Development" could be used to  
25 identify suitable tasks and pair constellations to ensure that the task is manageable for the novice and  
26 that developer skill levels are not too far apart.  
27  
28  
29  
30  
31  
32  
33

## 34 **5.2 RQ2: In which ways do the roles of driver and navigator influence knowledge transfer** 35 **in pair programming?** 36 37 38 39 40

41 Experts adapt their teaching strategies according to their role: they engage the novice through  
42 explanations and verbalisations while driving and guide the novice with instructions while navigating.  
43  
44  
45

46 Novices' behaviours are also influenced by whether they are driving or navigating.  
47  
48

49 Novices become more active and engage on a more detailed level when driving than they do when  
50 navigating. Novices are encouraged to think through, understand and solve parts of the problems by  
51 themselves when using strategies (1)-(3) and to perform the necessary steps by following the expert's  
52 instructions when using strategy (4). This means that novices "learn by doing" when driving rather than  
53 observing. Example 11 shows that the novice asked more questions as soon as the expert suggested a  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

1 role switch, thus seeming to engage more with the task. In the interviews, experts and novices agreed  
2 that if the novice drives it is beneficial for the novice learning but acknowledged that having the novice  
3 drive takes more time to solve the task than when the expert drives. This is in line with existing research  
4 that an “extremely effective style of learning by doing” occurs when learners solve as much of the work  
5 as possible guided by a tutor [22].  
6  
7  
8  
9  
10

11 The novice’s behaviour in Example 11 indicates that novices hesitate when it is their turn to drive (in  
12 line with the findings in [29]) even though it is supposed to be beneficial for their learning process.  
13 Plonka et al. [28] showed that often the expert dominates the driving in expert–novice constellations.  
14 This emphasises how important it is for the expert to encourage role switches, and for the novice to be  
15 prepared to drive.  
16  
17  
18  
19  
20  
21  
22  
23  
24

### 25 **5.3 RQ3: What challenges do developers with different knowledge levels face when** 26 **pairing together** 27 28

29 Expert–novice constellations provide learning opportunities for both partners, and experts also face  
30 challenges working with a novice. We do not report on challenges for the novice in this paper, but  
31 novice challenges such as social pressure are described in [29].  
32  
33  
34  
35  
36  
37  
38  
39

#### 40 **5.3.1 Opportunities for expert learning** 41

42 Expert–novice constellations focus mainly on knowledge transfer from the expert to the novice, but this  
43 research identified learning opportunities for experts as well. Novices are usually less familiar with the  
44 code than the expert, so they can provide a different perspective on the problem and existing code.  
45 Novices might ask “simple questions” (see Example 7) that force the expert to rethink previously-held  
46 ideas thereby uncovering problems and leading to code improvements. The novice might also suggest  
47 solutions that an expert had not seen or considered (see Example 13).  
48  
49  
50  
51  
52  
53  
54  
55  
56

57 Novices have a “beginner’s mind”, which refers to people who are unfamiliar with a situation and who  
58 might consider more possibilities than an expert. Belshee [3] points out the positive effect of the  
59  
60  
61  
62  
63  
64  
65

1 beginner's mind on knowledge transfer in PP and in the context of requirements engineering. Berry [4]  
2 points out that novices might ask questions that help to expose assumptions by experts.  
3  
4

5 In the context of expert–novice constellations, this means that experts can benefit and learn from the  
6 fresh perspective of the novice and that simple questions might help the expert to challenge their own  
7 assumptions about the existing code.  
8  
9

10  
11  
12 Another potential for expert learning is the process of verbalisation; talking about a problem might lead  
13 to a better understanding of the problem itself [8]. In the interviews, experts stated that explaining their  
14 thoughts to the novice helped them to structure their thoughts and think them through more thoroughly.  
15  
16 Cao and Xu [7] also found that the process of verbalisation was helpful for the expert in order to readjust  
17 goals and reorganise thoughts even when the novice did not react to the explanation.  
18  
19  
20  
21  
22  
23  
24  
25

### 26 **5.3.2 Expert challenges**

27

28 On the other hand, explaining and verbalising can be an additional cognitive effort for the experts. In  
29 Example 9, an expert was seen to struggle while working through a complex problem. The expert was  
30 using verbalisation to assist himself to work through the problem. It was challenging because the expert  
31 was verbalising in order to explain the code to the novice, and so the novice was free to ask questions  
32 and make comments during this time. The expert was not able to deal with both his verbalisation and the  
33 novice's comments and questions. This stresses the importance of being aware that developers might  
34 verbalise to structure their own thoughts and hence might not be able to react to their partner's  
35 comments until they have finished this process.  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47

48 PP has been described by some as an exhausting practice [41, 37]. While [41] ascribes this to developers  
49 focusing more on the task due to the pair pressure, our research shows that the effort of verbalization  
50 and explanation could be another reason why PP is more tiring than solo programming.  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

## **6 Limitations**

Whilst great care was taken in designing and conducting the study, we acknowledge that some issues in the data gathering, research method and the analysis could be viewed as limitations of the research. In particular we identify the following issues: Voluntary participation (potentially affecting completeness), developers being observed and recorded (potentially affecting generalizability), and detailed qualitative analysis based on a relatively small amount of data (potentially affecting completeness and generalizability). Note that none of these limitations affect the validity of the findings.

### **6.1 Voluntary participation**

Participation was voluntary for all developers. That is, developers from the identified companies were invited to participate, but no pressure was used to attempt to get all developers from each company to be involved. This means that developers who are confident in their pair programming skills or have a positive attitude towards pair programming might have been more likely to participate. This in turn may have affected the findings because less confident developers or developers with a negative attitude towards pair programming might exhibit different behaviours. This limitation potentially affects the completeness of our results.

### **6.2 Developers being observed and recorded**

Video and audio data were gathered during the pair programming sessions. Developers were informed about the recording before they started their sessions and it is possible that they modified their behaviour due to the fact that they knew they were being studied. To minimise the effect of participants feeling observed the recording setup was integrated in the developers workplace (using a webcam and wireless microphones that do not restrict any kind of movement, for example getting up from the chair). Additionally, in the interviews, developers were asked whether they had felt conscious of being observed. Some developers stated that they were aware of the webcam, others stated that they

1 completely forgot about it once they started working on the task. Some developers even replied that they  
2 felt more observed by their partner than by our recording setup. This last statement suggests that pair  
3 programming sessions are situations in which developers might feel observed even without being  
4 recorded. This limitation potentially affects the generalisability of our results.  
5  
6  
7  
8  
9

### 10 **6.3 Detailed qualitative analysis based on a small amount of data**

11  
12 In this paper, we used a detailed and qualitative analysis approach rather than a quantitative perspective  
13 on knowledge transfer in PP. This means that the very time-consuming steps (for example group  
14 viewing) of the interaction analysis were conducted on only a small subset of the overall data. However,  
15  
16 all 21 sessions had been analysed previously to ensure that the excerpts present typical pair  
17 programming episodes. Therefore, these findings, while not generalisable, do capture a good degree of  
18  
19 the variability faced within pair programming sessions.  
20  
21  
22  
23  
24  
25  
26  
27  
28

## 29 **7 Conclusions and Future Work**

30  
31  
32 So, what does it take to be a good “expert” and to learn best as a “novice”?

33  
34 We have shown that developers, without any explicit training, use strategies to transfer knowledge  
35 between expert and novice developers that are examples of some of the teaching methods used in  
36 cognitive apprenticeship – in particular using forms of Modeling, where the expert verbalizes their  
37 thought process, and Coaching and Scaffolding where novice developers are supported while they take  
38 an active part in the task. Not all of the teaching methods from cognitive apprenticeship were seen, and  
39 two of them (Reflection and Exploration) may not be easily transferrable to the specific context of agile  
40 software practice, where business value is typically prioritised over training needs, and the main focus is  
41 on producing code ready for integration. However, novice Articulation, where a novice verbalizes their  
42 own thought process, was not encouraged, yet this would enhance the novice’s learning experience. Not  
43 only was novice Articulation not encouraged in any of the exemplars we analysed in detail, we can also  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60 claim, due to step 1 in our procedure, that it did not feature in any of the sessions we recorded.  
61  
62  
63  
64  
65

1 Although experts need to make an effort to transfer knowledge, they regard sessions in which novices  
2 ask questions as rewarding since it helps them to reflect on their own practices and thus learn from the  
3 experience. An increased awareness of working practices for knowledge transfer in PP will help  
4 developers to maximise the benefits from such sessions, and provide learning opportunities even for the  
5 expert.  
6  
7  
8  
9  
10

11 This study focused on expert–novice constellations in order to highlight knowledge transfer activities,  
12 but a certain amount of knowledge transfer takes place in all PP sessions. One future direction will be to  
13 investigate which (if any) of these findings are evident in other PP sessions where knowledge transfer is  
14 not the main purpose.  
15  
16  
17  
18  
19  
20  
21

## 22 **Acknowledgements**

23 We would like to thank all the developers and companies who made this research possible by  
24 participating in the studies. "  
25  
26  
27  
28  
29  
30  
31

## 32 **References**

- 33  
34  
35  
36 [1] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 1999.  
37  
38 [2] A. Begel and N. Nagappan. Pair programming: what's in it for me? In *ESEM '08: Proceedings of*  
39 *the Second ACM-IEEE international symposium on Empirical Software Engineering and*  
40 *Measurement*, pages 120–128. ACM, 2008.  
41  
42  
43  
44 [3] A. Belshee. Promiscuous pairing and beginner's mind: embrace inexperience. In *Proceedings of the*  
45 *Agile Development Conference, ADC '05*, pages 125–131, 2005.  
46  
47  
48  
49  
50  
51 [4] D. M. Berry. The importance of ignorance in requirements engineering. In *Journal of Systems and*  
52 *Software*, 28, pages 179–184. Elsevier, 1995.  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65



- 1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65
- [5] K. Børte, S. R. Ludvigsen, and A. I. Mørch. The role of social interaction in software effort estimation: Unpacking the “magic step” between reasoning and decision-making. *Information and Software Technology*, 54(9), 985–996, Sept. 2012.
  - [6] F. Brooks. *The mythical man-month*. Addison-Wesley Reading, Mass, 1975.
  - [7] L. Cao and P. Xu. Activity patterns of pair programming. In *HICSS’05. Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, pages 88a–88a. IEEE, 2005.
  - [8] M. Chi, M. Bassok, M. Lewis, P. Reimann, and R. Glaser. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive science*, 13(2),145–182, 1989.
  - [9] J. Chong and T. Hurlbutt. The social dynamics of pair programming. In *Proceedings of the 29th International Conference on Software Engineering, ICSE ’07*, pages 354–363, 2007.
  - [10] A. Cockburn and L. Williams. The costs and benefits of pair programming. In *Extreme Programming Examined*, pages 223–248, Addison-Wesley, 2001.
  - [11] A. Collins, J. Brown, and A. Holum. *Cognitive apprenticeship: Making thinking visible*. volume 15, pages 6–11. American Federation of Teachers, 1991.
  - [12] A. Collins, J. S. Brown, and S. Newman. *Cognitive apprenticeship: Teaching the craft of reading, writing, and mathematics*. Technical report no. 403, BBN Laboratories, Cambridge, MA. Centre for the Study of Reading, University of Illinois, January 1987.
  - [13] V. Dennen. Cognitive apprenticeship in educational practice: Research on scaffolding, modeling, mentoring, and coaching as instructional strategies. *Handbook of research on educational communications and technology*, 2, 813–828, 2004.
  - [14] P. Dillenbourg. What do you mean by collaborative learning? In *Collaborative Learning: Cognitive and Computational Approaches*, pages 1–19. Emerald Group Publishing Limited, 1999.

- 1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65
- [15] Y. Dittrich and R. Giuffrida. Exploring the role of instant messaging in a global software development project. In 6th IEEE International Conference on Global Software Engineering (ICGSE), pages 103–112, 2011.
  - [16] B. Greene. Agile methods applied to embedded firmware development. In Proceedings of the Agile Development Conference, pages 71–77, 2004.
  - [17] P. Hodgetts. Refactoring the development process: Experiences with the incremental adoption of agile practices. In Proceedings of the Agile Development Conference, ADC '04, pages 106–113, Washington, DC, USA, IEEE Computer Society, 2004.
  - [18] R. Jensen. A pair programming experience. *CrossTalk: A Journal of Defense Software Engineering*, 16(3), 22–24, 2003.
  - [19] B. Jordan and A. Henderson. Interaction analysis: Foundations and practice. *The Journal of the Learning Sciences*, 4(1), 39–103, 1995.
  - [20] S. Katriou and E. Tolia. From twin training to pair programming. In Proceedings of the 2nd India software engineering conference, ISEC '09, pages 101–104, New York, NY, USA, 2009. ACM.
  - [21] G. Luck. Subclassing xp: Breaking its rules the right way. In Proceedings of the Agile Development Conference, ADC '04, pages 114–119, 2004.
  - [22] D. Merrill, B. Reiser, S. Merrill, and S. Landes. Tutoring: Guided learning by doing. *Cognition and Instruction*, 13(3):pp. 315–372, 1995.
  - [23] A. Pandey, C. Miklos, M. Paul, N. Kameli, F. Boudigou, V. Vijay, A. Eapen, I. Sutedjo, and W. Mcdermott. Application of tightly coupled engineering team for development of test automation software - a real world experience. In Proceedings of the 27th Annual International Conference on Computer Software and Applications (COMPSAC '03), page 56, Washington, DC, USA, IEEE Computer Society, 2003.

- 1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65
- [24] N. Phaphoom, A. Sillitti, and G. Succi. Pair programming and software defects – an industrial case study. In *Agile Processes in Software Engineering and Extreme Programming*, volume 77 of *Lecture Notes in Business Information Processing*, pages 208–222. Springer, 2011.
  - [25] M. Phongpaibul and B. Boehm. An empirical comparison between pair development and software inspection in Thailand. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, ISESE '06*, pages 85–94, New York, NY, USA, ACM, 2006.
  - [26] L. Plonka. *Unpacking Collaboration in Pair Programming in Industrial Settings*. PhD thesis, The Open University, 2012.
  - [27] L. Plonka, J. Segal, H. Sharp, and J. Linden. Collaboration in pair programming: Driving and switching. In A. Sillitti, O. Hazzan, E. Bache, and X. Albaladejo, editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 77 of *Lecture Notes in Business Information Processing*, pages 43–59. Springer, 2011.
  - [28] L. Plonka, J. Segal, H. Sharp, and J. van der Linden. Investigating equity of participation in pair programming. In *Proceedings of the 2012 Agile India, AGILEINDIA '12*, pages 20–29, Washington, DC, USA. IEEE Computer Society, 2012.
  - [29] L. Plonka, H. Sharp, and J. van der Linden. Disengagement in pair programming: Does it matter? In *34th International Conference on Software Engineering (ICSE)*, pages 496–506, 2012.
  - [30] J. Roschelle and W. Clancey. Learning as social and neural. *Educational Psychologist*, 27(4), 435–453, 1992.
  - [31] S. Salinger, F. Zieris, and L. Prechelt. Liberating Pair Programming Research from the Oppressive Driver/Observer Regime. In *35th International Conference on Software Engineering (ICSE)*, pages 1201–1204, 2013.
  - [32] D. Sanders. Student perceptions of the suitability of extreme and pair programming. In *Extreme Programming Perspectives*, Chapter 23, Addison Wesley, 2002.

- 1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65
- [33] C. Schindler. Agile software development methods and practices in Austrian IT-industry: Results of an empirical study. In Proceedings of the 2008 International Conference on Computational Intelligence for Modelling Control & Automation, CIMCA '08, pages 321–326, Washington, DC, USA. IEEE Computer Society, 2008.
- [34] T. VanDeGrift. Coupling pair programming and writing: learning about students' perceptions and processes. In Proceedings of the 35th SIGCSE technical symposium on Computer science education, volume 36 of SIGCSE'04, pages 2–6, New York, NY, USA. ACM, Mar. 2004.
- [35] J. Vanhanen and H. 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, Washington, DC, USA. IEEE Computer Society, 2007.
- [36] J. Vanhanen and C. Lassenius. Effects of pair programming at the development team level: an experiment. In Proceedings of the 2005 International Symposium on Empirical Software Engineering, page 10, 2005.
- [37] J. Vanhanen and C. Lassenius. Perceived effects of pair programming in an industrial context. In Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, pages 211–218, 2007.
- [38] J. Vanhanen, C. Lassenius, and M. Mantyla. Issues and tactics when adopting pair programming: A longitudinal case study. In Proceedings of the International Conference on Software Engineering Advances, ICSEA '07, page 70. IEEE Computer Society, 2007.
- [39] L. Vygotsky. Mind in Society: The development of higher psychology processes. Cambridge MA: Harvard University press, 1978.
- [40] L. Williams. But, isn't that cheating? In Frontiers in Education Conference, 1999. FIE'99. 29th Annual Conference, volume 2, pages 12B9–26. IEEE, 1999.

1 [41] L. Williams and R. Kessler. Pair programming illuminated. Addison-Wesley Longman Publishing  
2 Co., Boston, MA, USA, 2002.  
3

4 [42] L. Williams, A. Shukla, and A. Anton. An initial exploration of the relationship between pair  
5 programming and Brooks' law. Agile Development Conference, pages 11–20, 2004.  
6  
7  
8  
9

10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65