Peer-Based Quality Assurance in Information Systems Development: A Transactive Memory Perspective
Completed Research Paper

Kai Spohrer
University of Mannheim
Mannheim, Germany
spohrer@uni-mannheim.de

Thomas Kude
University of Mannheim
Mannheim, Germany
kude@uni-mannheim.de

Armin Heinzl
University of Mannheim
Mannheim, Germany
heinzl@uni-mannheim.de

Christoph T. Schmidt
SAP AG &
University of Mannheim
Walldorf and Mannheim, Germany
christoph.schmidt@uni-mannheim.de

Abstract

This study provides a better understanding of how the application of collaborative software development techniques is shaped by and shapes the transactive memory systems of information systems development teams. In particular, we draw from results of critical realist case studies on development teams that apply pair programming and peer code review. The findings improve our understanding of how teams vary in the way they apply these collaborative quality assurance techniques and how their application is theoretically linked to a team’s transactive memory system, as well as the properties of the technologies the techniques rely on. This study replies to recent calls to study socio-cognitive implications of software development techniques. It contributes to previous work on quality assurance techniques by being among the first to empirically study effects that go beyond error correction. This also adds substance to the discussion if different techniques complement or substitute each other.

Keywords: Software development, transactive memory systems, functional affordances, pair programming, peer code review
Introduction

Routines and roles reduce cognitive load on individuals and allow for division of work (Feldman & Rafaeli, 2002). In traditional information systems development (ISD), such roles included dedicated testers and quality managers, while teams of developers were primarily responsible for creating new code. Contemporary ISD, based on the ideas of agile software development, however, considers teams of empowered developers as the decisive entities not only for developing code, but also regarding quality. Consequently, agile ISD teams have accommodated iterative, collaboration-based quality assurance techniques in their daily routine (Erickson et al., 2005).

Given the knowledge-intensive nature of ISD and the entailing cognitive complexity, the question arises what such new techniques mean for the joint cognitive accomplishment of software development teams (Davern et al., 2012). On the one hand, these techniques may require increased cognitive effort from developers, thus potentially decreasing their performance within the team. On the other hand, the associated activities may provide developers with opportunities to learn about their teammates, thereby strengthening the team’s socio-cognitive structure, that is, the thinking and knowledge that extends beyond the individual (Davern et al., 2012). In this paper, we examine peer-based quality assurance techniques, defined as ISD practices in which single developers on the same hierarchical level and members of the same team ensure each other’s code quality in an ongoing manner during daily routine. In particular, we focus on pair programming and peer code review and their interplay with the transactive memory system of ISD teams. Transactive memory systems is a promising theoretical lens to examine this phenomenon because it captures both the effects of individuals’ knowledge as well as their interactions on group-level achievements.

Pair programming (PP) is an ISD technique in which two developers work collaboratively on the same software fragment, located at a single computer (Balijepally et al., 2009). Alternately, one of them takes the role of a driver who actively writes source code using the hardware, whereas the other is “observing the work of the driver and identifying tactical and strategic deficiencies in their work” (Williams, 2000, p. 3). Pair programming is by far the most widely adopted agile software development technique (Balijepally et al., 2009).

When using peer code reviews (PCR), source code is developed by one software engineer, the author, who submits the code to an electronic review system. Fellow developers then examine the code for its quality and correctness, propose changes, highlight defects, and recommend improvements (Rigby et al., 2012). For many years, PCR has been a central quality assurance technique in open-source software communities (Rigby & Storey, 2011; Liang & Mizuno, 2011). It is currently diffusing to ISD companies (Rigby et al., 2012) and has been proposed as a less costly alternative to PP in practice (Paulk, 2001; Rigby et al., 2012). In fact, both techniques come at a considerable expense of resources invested in observing or reviewing (Arisholm et al., 2007; Balijepally et al., 2009). Therefore, it appears crucial for ISD teams to understand the preconditions and consequences of applying peer-based quality assurance techniques in order to effectively use available capacity.

Peer-based quality assurance techniques have not only been suggested to increase code quality but also as mechanisms for knowledge exchange (Rigby et al., 2012). In fact, previous research found PP to be particularly useful when pairs were required to complete complex tasks (Balijepally et al., 2009; Williams, 2000; Salleh et al., 2011). However, this existing body of literature has two major shortcomings: first, studies have focused primarily on efforts and quality gains in single development tasks. While they acknowledged the existence of socio-cognitive effects between the actors, research has yet to unveil the nature and origin of these effects. Second, scholars have not addressed the questions if or how such socio-cognitive effects on dyads of developers have consequences for larger ISD teams in which they are embedded. This is crucial, however, as contemporary ISD work within organizations is not only knowledge-intensive, but most often also distributed within teams rather than pairs (Janz & Prasarnphanich, 2009). A transactive memory perspective sheds light on this area by detailing the effects of individual knowledge and interactions on group-level efficiency. Against this backdrop, we seek to answer the following research question:

*How and why do peer-based quality assurance techniques interact with the transactive memory system of information systems development teams?*
Answers to this question promise important contributions to theory and practice. By examining peer-based quality assurance and its interplay with the transactive memory of teams, this study replies to recent calls for research on socio-cognitive effects of ISD techniques (Davern et al., 2012). In doing so, it also contributes to theory in explaining how two ISD techniques that are performed between single team members result in effects on the overall ISD team. Despite the common focus on software development teams instead of individuals, prior research has largely disregarded this perspective (Sarker et al., 2011). Moreover, this study is one of the first to examine both PP and PCR in collocated ISD teams within software companies, thereby investigating team-level phenomena not accessible to prior work that was mostly confined to laboratory settings or open-source communities (Schmidt et al., 2012). Finally, this study shows that ISD teams can use these techniques as complements if their transactive memory is sufficiently sophisticated and the techniques' specific technological properties are leveraged. This work, thereby, lays the foundation for future research on how and why stable patterns emerge within ISD teams regarding the use of peer-based quality assurance techniques as well as research on the implications for ISD team performance.

The paper is structured as follows. The next section provides an overview of extant work on peer-based quality assurance and socio-cognitive phenomena in ISD teams, thus reemphasizing the identified void that this study seeks to address. Subsequently, we discuss transactive memory systems (TMSs) that provide the theoretical foundation of our study. Then, the methodological approach of our work, a critical realist case study, is motivated and introduced by presenting the data selection and analysis procedures. Next, the results of this case study are presented by providing insights into the observed activities and relating these events to the transactive memory of the respective teams. Finally, we discuss our findings in terms of contributions to the existing body of literature as well as implications for ISD teams in practice.

Literature Review and Theoretical Foundation

Peer-Based Quality Assurance

Pair programming and peer code review represent two collaborative ISD techniques that share the goal of ensuring high code quality through the interaction of two developers during software development. There are many quality assurance techniques in ISD, ranging from approaches that can be executed by single developers, such as unit tests or checklist-based evaluations, to post-implementation code inspections by entire ISD teams (Fagan, 1986). We focus on PP and PCR because these techniques share two distinctive characteristics compared to other approaches (Williams, 2000; Rigby et al., 2012): (i) they rely on a combination of single developers from one team to jointly produce better code and (ii) they are executed during implementation, that is, before the source code leaves the ISD team to be integrated into a software product which may be tested by external persons. These techniques therefore depend on the distribution of knowledge across different team members, but also imply interactions to retrieve or alter such knowledge distributions, which makes them particularly interesting for studies of transactive memory systems.

Pair programming has drawn attention in the fields of information systems and software engineering (Dingsøyr et al., 2012; Balijepally et al., 2009; Erickson et al., 2005) as well as in education (Salleh et al., 2011). Existing work on PP is primarily concerned with evaluating the effectiveness and efficiency of pairing developers. The primary point of reference for such evaluations has been solo programming of single developers (for reviews see Salleh et al., 2011; Dybå & Dingsøyr, 2008; Hannay et al., 2009; Dingsøyr et al., 2012). Beyond efficiency arguments, PP has been shown to change the knowledge distribution between pairing developers, emphasizing learning effects of less experienced from more experienced peers (Salleh et al., 2011). At the same time, however, scholars found that differences in knowledge and skills between two developers not only change during PP, but may also determine success or failure of common PP sessions: too small differences may prevent learning effects while too large differences may prevent effective collaboration during PP (Salleh et al., 2011). It has been suggested that junior developers might consequently gain more from PP than experienced developers who should therefore refrain from PP altogether (Dybå et al., 2007). Only very few studies, however, investigated PP of dyads within larger ISD teams (cf. Salleh et al., 2011; Dybå & Dingsøyr, 2008; Dingsøyr et al., 2012). Generally, findings on team-level effects do not extend beyond mere
acknowledgements that learning effects should exist (Vidgen & Wang, 2009). Despite the fact that ISD rarely takes place in isolated dyads but rather in ISD teams (Janz & Prasarnphanich, 2009), investigations into PP in larger teams are lacking (Davern et al., 2012). Such studies may not only help to better understand the effects of PP beyond single PP sessions, but also shed light on how ISD teams utilize the knowledge of their members.

In contrast to PP, PCR received only little attention in research even though frequent, iterative peer code reviews of small pieces of code have long been used as a central quality assurance technique in open-source communities (Rigby et al., 2012). Existing work on PCR was primarily concerned with its mechanisms for error detection in open-source communities (Rigby et al., 2008; Rigby & Storey, 2011; Liang & Mizuno, 2011), whereas little is known about the application of PCR in collocated teams within for-profit organizations or its link to socio-cognitive aspects. Given the characteristic similarities of PP and PCR, surprisingly few attempts have been made to contrast one with the other (Müller, 2004, 2005; Rigby et al., 2012; Schmidt et al., 2012). In particular, technology-mediated PCR has remained almost disregarded (Rigby et al., 2012; Schmidt et al., 2012), although findings on non-technology-based code reviews suggest that it is the direct and intensive social interactions which drives costs in this type of code review similar to PP (Müller, 2005). There is good reason to believe that such interactions may be less intensive and possibly less costly in PCR (Rigby et al., 2012; Schmidt et al., 2012). Finally, prior research suggests that technology can significantly alter the development and use of a transactive memory in teams to the better or the worse (Hollingshead, 1998a; Choi et al., 2010; Jarvenpaa & Majchrzak, 2008). Examining two comparable quality assurance techniques that differ in the technology involved, therefore, also promises to increase our understanding of the role of technology in forming ISD teams’ transactive memory.

**Transactive Memory Systems**

Recent reviews of IS literature highlight that ISD research would benefit from an extended understanding of socio-cognitive phenomena (Davern et al., 2012; Spohrer et al., 2012). The core premise of such phenomena is that, in order to complete the interdependent tasks of teams, individual team members interact through sharing and complementing each other’s cognitive resources (for selected reviews see Cannon-Bowers & Salas, 2001; Ilgen et al., 2005; Edmondson et al., 2007; Wilson et al., 2007; Ren & Argote, 2011; Lewis & Herndon, 2011). Extant literature on the socio-cognitive structure of ISD teams has broadly been classified based on the conceptualizations of the phenomena it focuses on as well as theories, methods, and contributions (cf. Spohrer et al., 2012): in essence, the best-established stream of literature addresses ISD teams’ transactive memory systems and investigates how they acquire, store, and use knowledge (e.g., Faraj & Sproull, 2000; He et al., 2007; Oshri et al., 2008; Maruping et al., 2009). However, such research usually does not incorporate individual actions in its analysis, which ignores the important influence of single developers on the teams they are embedded in (Sarker et al., 2011). Only recently, research has made first attempts to account in more detail for the role of individual developers in shaping the cognitive structures of entire ISD teams (Sarker et al., 2011; Skerlavaj et al., 2010). Our study adds to this literature by examining ISD techniques that are carried out by single developers but are embedded in and yield effects on ISD teams’ transactive memory.

This paper focuses on ISD teams of collocated individual developers that interact to jointly produce software. As a result of this close interaction, such a collocated team may develop a transactive memory structure that allows for utilizing distributed knowledge (Wegner, 1987; Hollingshead, 1998a). In particular, teams may develop a TMS as a form of social cognition through which they distribute and retrieve specialized knowledge from single team members (Wegner, 1987; Choi et al., 2010). A team’s TMS develops when specialized knowledge held by single team members becomes available to the rest of the team as they associate expertise in an area with the respective person and understand which team tasks require that expertise (Brandon & Hollingshead, 2004). By means of a TMS, team members are therefore enabled to engage in division of cognitive work and thus exceed the performance of isolated individuals (Wegner, 1987). Indeed, teams with well-developed TMS were found to show better recall of information as well as work performance (Liang et al., 1995; Hollingshead, 1998a; Moreland & Myaskovsky, 2000; Kanawattanachai & Yoo, 2007; Majchrzak et al., 2007; Jarvenpaa & Majchrzak, 2008). Technology can affect TMS development and use positively.
as well as negatively: on the one hand, technology can support communication and TMS use in distributed
teams (Kanawattanachai & Yoo, 2007), but it may hinder direct personal communication in collocated teams
which would be beneficial for TMS creation and use (Hollingshead, 1998a); on the other hand technology
can support the codification of who knows what in a team and thereby stimulate TMS development and use
(Choi et al., 2010; Nevo & Wand, 2005; Kotlarsky et al., 2013). Examining PP with its focus on personal
interactions and PCR with its technology-mediated communication both in collocated ISD teams therefore
promises interesting insights for the role of technology in TMS creation and utilization.

TMS is recognized as a powerful concept and frequently applied in IS, management and organizational re-
search (for selected reviews see Ilgen et al., 2005; Ren & Argote, 2011; Lewis & Herndon, 2011; Nevo &
Ophir, 2012). In particular, TMSs are considered to be highly relevant for ISD teams and the outcome of
their collaborative work (Faraj & Sproull, 2000; He et al., 2007; Oshri et al., 2008; Kotlarsky & Oshri, 2005;
Lin et al., 2011). Interestingly, the influence of actual ISD team work activities on ISD teams’ TMS has been
studied only scarcely (Spohrer et al., 2012), although existing work indicates that there are effects that may
partially determine an ISD team’s performance (Maruping et al., 2009). Transactive memory systems con-
sist of two components: (i) TMS structure, that is an organized storage of distributed expert knowledge as
well as commonly shared meta-knowledge; and (ii) transactive processes, i.e., human activities to encode,
store and retrieve the knowledge (Rulke & Rau, 2000; Ren & Argote, 2011). These two components are
assumed to be closely intertwined, “with TMS structure influencing the nature and efficiency of transactive
encoding, storage and retrieval processes, and those same TMS processes in turn updating and refining the
TMS structure” (Lewis & Herndon, 2011, p. 1256). This tight entanglement between TMS structure and
TMS processes has implications for the conceptualization of TMS. In fact, when indirectly assessing TMS,
previous literature has suggested to infer its existence from the simultaneous assessment of the knowledge
specialization within a team, the credibility that team members assign to others’ knowledge, as well as the
ability of team members to coordinate distributed knowledge (Lewis, 2003; Lewis & Herndon, 2011). Thus,
this indirect conceptualization of TMS assesses both the team’s TMS structure and its transactive processes
in a simultaneous and inseparable way.

As opposed to such an indirect approach, there is also previous literature that conceptualized TMS by ana-
lytically and empirically separating TMS structure and the cognitive processes used to apply and alter this
structure (Lewis et al., 2007; Rulke & Rau, 2000; Hollingshead, 1998a,b). We subscribe to this latter per-
spective and see TMS structure as limitedly observable but possessing the power to influence observable
behaviors. In the context of this study, behavior refers to the observable events of two or more team mem-
ers applying PP or PCR. The key premise of our study is that when applying these quality assurance tech-
niques, individuals execute transactive processes. That is, individuals assign labels to knowledge (encoding),
archive knowledge or meta-knowledge (storage), and access previously stored knowledge (retrieval) (Lewis
et al., 2007; Hollingshead, 2001). Whereas the observability of a team’s TMS structure is limited, the events
of applying PP or PCR may entail observable changes in the individuals’ knowledge and meta-knowledge.
Such small, observable changes on an individual level may emerge to changes in the team-level TMS struc-
ture (Klein & Kozlowski, 2000). Generally, TMS structure improves if shared meta-knowledge about new or
existing specialized knowledge is developed and mapped to the matching tasks (Brandon & Hollingshead,
2004). For instance, one team member may develop specialized knowledge about how to query a specific
data base. Then, the associated meta-knowledge could be, for example, knowledge about who possesses the
knowledge to query the data base, what this knowledge can be used for, or if the knowledge of this person
is credible (Nevo & Wand, 2005). If such meta-knowledge is shared among team members, an increase in
team members’ knowledge and meta-knowledge leads to an evolving TMS structure on a team-level.

**Research Design**

Previous empirical studies on peer-based quality techniques have mostly been restricted to laboratory set-
tings or open source communities (Schmidt et al., 2012). In order to complement these research efforts,
we sought to examine the interplay of these techniques with a team’s TMS within collocated ISD teams of a
software company. As a consequence, we had no immediate control over the unfolding of the phenomena of
interest. Therefore, a qualitative case study approach was deemed particularly appropriate as it promised rich insights and the flexibility to adapt our investigation in an ongoing manner as we deepened our own understanding (Wynn & Williams, 2012).

Our research takes the epistemological and ontological stance of critical realism (Archer et al., 1998; Bhaskar, 1998; Mingers, 2004). Based on recent guidelines consistent with this stance (Wynn & Williams, 2012), we conducted extensive case study research within a large, global software development company. This organization was particularly interesting for our research as its ISD teams are working based on the agile ISD method Scrum (Schwaber & Beedle, 2002). Moreover, the teams within this organization have complete discretion if and how to apply specific ISD techniques. Many of the teams received trainings in agile ISD techniques, including PCR and PP, within the scope of an organizational transformation toward agile ISD during winter 2011/2012.

**Data Collection**

Our qualitative investigation consisted of three phases: we (i) elicited the background the organization provided to its ISD teams, (ii) broadly explored the role and use of PP and PCR within the company, and (iii) engaged in four in-depth case studies (cf. Figure 1). In order to understand the background of the ISD teams in the case company, one of us immersed in the organization over a one year period, visiting the field site several times a week. These visits included attending the standard trainings in agile software development provided to many of the ISD teams in the company, eliciting opinions of developers about the different techniques they were trained in, and playing these insights back to the providers of the trainings. In addition, two of the authors jointly attended a special training in peer-based quality assurance during summer 2012. By means of this close embeddedness into the case company, we were enabled to gain profound knowledge about the trainings and the conveyed understanding of the techniques, as well as about the role of the techniques within the organization. Moreover, we gained valuable insights into the organizational structure and culture of the case company.

We further explored the role of peer-based quality assurance in terms of the situation in which it was applied,
the way the teams used the techniques, and which goals were pursued by doing so: based on a total number of 13 interviews with key informants such as agile method trainers, Scrum masters, and senior developers, peer-based quality assurance was explored in a more systematic way within nine ISD teams. On average, these interviews lasted approximately one hour. All interviews were tape-recorded and transcribed. Eight of the studied teams applied one or both techniques while one team (REVIEW) used - but also actively developed - an open-source PCR platform. This exploratory phase provided us with a comprehensive picture of very different teams in different contexts. Table 1 provides an overview of this exploration.

<table>
<thead>
<tr>
<th>PP Use</th>
<th>INSTALL3</th>
<th>UI</th>
<th>CORRECT</th>
<th>BPM1</th>
<th>INSTALL1</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>high</td>
<td>high</td>
<td>high</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>no</td>
<td>Version Verification for Products</td>
<td>low</td>
<td>moderate</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>INSTALL2</td>
<td>PORTAL</td>
<td>REVIEW</td>
<td>COMPILE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PP Use</td>
<td>moderate</td>
<td>low</td>
<td>low</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>moderate</td>
<td>high</td>
<td>high</td>
<td>high</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client Installation Suite</td>
<td>Web Portal</td>
<td>PCR Tool</td>
<td>Custom Language Compiler</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team Size</td>
<td>3</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Interviews</td>
<td>SD</td>
<td>SM/TR</td>
<td>SD</td>
<td>SM/TR, SD</td>
<td>SD, SD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PP Use</th>
<th>COMPILE</th>
<th>BPM</th>
<th>DELIVER</th>
<th>SELL</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>high</td>
<td>high</td>
<td>moderate</td>
<td>no</td>
</tr>
<tr>
<td>high</td>
<td>high</td>
<td>moderate</td>
<td>Development Infrastructure Software</td>
<td>Online Sales Platform</td>
</tr>
<tr>
<td>Team Size</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Equipped with these insights and the necessary knowledge and understanding of the organizational context, we conducted in-depth case studies of four ISD teams (Wynn & Williams, 2012). The four ISD teams were selected purposefully regarding the peer-based quality assurance techniques they used and the frequency of application. As such, two of these teams used both PP and PCR while the others relied on either PP or PCR, respectively. Table 2 summarizes the four teams.

We studied these four teams intensively in order to obtain a rich picture of single sessions of PP and PCR. More specifically, we gathered data in multiple ways from diverse sources: first, through 12 days of participatory observations of the teams’ daily routine, including their meetings, daily work, and during these days, approximately 30 PP and PCR sessions; second, through semi-structured interviews of 10 to 40 minutes with involved team members right after each of the 30 sessions in which they applied PP or PCR; third, through semi-structured interviews with the four Scrum masters of the teams that lasted 35 to 60 minutes and were conducted before the first observations in the respective teams; fourth, through informal interviews with two product owners and two managers of the teams; fifth, through archival data from the teams’ PCR systems including several hundred code reviews, and finally, through a multitude of informal talks with team members and stakeholders during occasions such as common coffee breaks and lunches.

The participatory observations were conducted by two researchers at a time in order to provide different viewpoints on PP and PCR sessions as well as the general collaborative and coordinative activities in the
teams. During the single occurrences of PP, we focused on the contribution of expertise and meta-knowledge by the single actors, how this was acknowledged by the partners and incorporated into task completion. Immediately after having quietly observed the developers interactions, that is after they had completed their task or closed the session for other reasons, we interviewed the respective team members. We did so in order to elicit why they had conducted PP, what expertise had been required for this task, who contributed in what way to the result, why exactly they had become involved in this particular task, and how they evaluated the result of PP in this case.

When team members engaged in PCR as authors or reviewers, we observed their interactions with the PCR systems and interviewed them separately after they had submitted a code change or review. Being granted full access to the PCR systems, we later matched the single observation with archival data of reviews and revisions in these systems. Moreover, we paid close attention during daily routine such as meetings and informal talks to any occasions where team members would refer to activities, challenges, or expertise related to PP or PCR sessions. As such, we observed multiple events in which developers informed the rest of their team briefly about task-related issues they had been struggling with during pair programming the day before, and sketched out roughly how they had solved these issues.

In addition, we conducted formal and informal interviews with team members and stakeholders such as product owners and managers in order to elicit information about the teams, their work context, and their perceived patterns of collaboration, specialization and knowledge coordination. Due to explicit requests, we recorded only selected interviews with stakeholders and team members electronically. Instead we relied heavily on field notes during both interviews and observations. Field notes and impressions were discussed at the end of the same day, and summaries were written and archived together with the notes for each full-day observation.

Data Analysis

Whereas the first phase of our study (immersion in organization, cf. Figure 1) had the goal of gaining a comprehensive understanding of the research context, data collected in the exploratory phase (Table 1) as well as during the four in-depth case studies (Table 2) was systematically analyzed. The first step of data analysis was to generally assess how, when, and to what end PP and PCR were applied within the case company. This was done by closely analyzing the transcripts of the 13 exploratory interviews with key individuals from nine ISD teams. In particular, we coded text fragments that contained statements relating to the situations in which the techniques are used, the conditions how they were applied, and the consequences or goals that were followed by applying the techniques (Miles & Huberman, 1994; Patton, 2002). This process resulted in a multitude of situations, conditions, and consequences that were identified and described with labels such as “complex tasks” (situation), “review intensity” (condition), “error correction”, or “knowledge transfer” (consequences). In an iterative process, we aggregated these labels by merging categories or dropping redundant ones in order to eventually come up with a limited number of patterns that describe the different forms how the teams used PP and PCR in a parsimonious way. These different patterns of use formed one important basis of the four in-depth critical realist case studies that were conducted next.

The four in-depth case studies, conducted in the third phase based on guidelines of Wynn & Williams (2012), resulted in a large amount of structured and unstructured data that we obtained from participatory observations, formal interviews, informal conversations, and from the teams’ PCR systems. We approached the challenge of analyzing this rich data by assigning pieces of information to one or more episodes of applying PP or PCR. We did so by writing stories for each of these episodes that integrated all the data available on this particular application of PP or PCR (observational, interviews, conversations, system data) and linked the episode to one of the identified patterns of use. In line with guidelines for critical realist case studies (Wynn & Williams, 2012), these stories represented events that were the primary objects of our analysis. We structured the events by describing the context of the episode (e.g., task, individuals involved), references to internal as well as shared knowledge and meta-knowledge, and indications for cognitive processes that retrieve or change this knowledge or meta-knowledge. We further grouped the single sessions of PP and PCR in all teams with regard to the goals they aimed at and way they were conducted. Contrasting these groupings with each other and the exploratory interviews gained in Phase 2, we arrived at a parsimonious
set of finer-grained use patterns that accurately described all sessions of PP and PCR that we observed or were told of in interviews.

In the context of this study, we aimed at explaining the interplay between team-level TMS structure and the patterns of use of peer-based quality assurance techniques on an individual level. As noted before, TMS structure can only be observed to a limited degree and a direct assessment of the distribution of specialized knowledge within real ISD teams was thus not feasible. Therefore, we elicited personal opinions from team members and Scrum masters if and how expertise was distributed and used in their teams. During observation days, we further had participants of single PP or PCR sessions reflect on their and others’ expertise in the sessions’ specific tasks and if it was related to their engagement in any way. Comparing and contrasting these statements not only within but also across the different cases (Miles & Huberman, 1994), we were able to outline relative assessments of transactive memory structures of the four teams. With TMS structures and use patterns at hand, we could then analyze how they interacted in the four teams. We did so with the help of “thought trials” (Wynn & Williams, 2012) that reflect tentative generative mechanisms with the potential to create use patterns. These thought trials were then iteratively probed by attempting to explain all available observations, which is an accepted procedure in cross-case analyses (Miles & Huberman, 1994, p. 271). This iterative process continued until the identified generative mechanisms consistently explained the interplay between team structure and patterns of applying PP and PCR.

**Empirical Results**

**Patterns of Using Peer-Based Quality Assurance Techniques**

The goal of the exploratory phase was to identify patterns of how the examined ISD teams were applying peer-based quality assurance techniques. The findings showed that the way how PP and PCR were used substantially varied across teams. That is, PCR was conducted differently in different teams, and so was PP. This was surprising given that the teams had attended similar trainings. Based on the exploratory interviews, we elicited the differences in the way how teams used the techniques as well as the motivations for doing so. Several informants stated that their teams used either PP or PCR in order to correct errors or to enforce conventions regarding source code style quite interchangeably. However, there were also teams that relied on one of the techniques for characteristics that the other could not provide. By some interviewees, PP was seen as more effective for evaluating not only the final quality of a solution but also the entire approach.

"Questions of design and solution approach are better discussed in pair programming. [...] I think I usually do have good ideas, but you always miss small nuances. That’s what a partner can bring in.” (Scrum master UI)

By contrast, PCR was seen as particularly useful for enhancing the resulting source code based on not only a single but multiple other perspectives:

"In the end, I set my focus [in PCR] on clean code. [...] Others watched for typos, my soft spot, others said formatting is important. [...] And of course, there are the functional things that you are interested in.” (Senior developer PORTAL)

Further analysis revealed that these different concepts of PP and PCR varied in two dimensions: the configuration of individuals involved in a PP or PCR session on the one hand, and the interactions and relations between these individuals on the other hand. Pair programming is restrictive regarding the configuration of ISD team members involved as there always has to be a pair of developers. However, these two developers, sitting at the same computer, are nearly unrestricted in terms of how they interact. They may silently follow their pair, exchange opinions, share knowledge, or challenge each other’s ideas. By contrast, PCR restricts interactions by forcing all participants to communicate through written text and primarily based on the source code under review. Regarding the configuration of actors involved in PCR as reviewers or passive readers, on the other hand, there are no such restrictions. The group of active reviewers for a single code change in PCR potentially includes the entire ISD team. Based on their involvement of multiple actors and
intensity of interaction between the peers, these nine teams’ approaches to PP and PCR could be grouped into four use patterns described in Figure 2.

These flexible properties of PP and PCR regarding participating actors and their interaction can be seen as embedded in the technological objects and artifacts that accompany them (Volkoff et al., 2007): it is due to collocation at the same computer during PP that actors can interact intensively, whereas their number is restricted; it is also a structural property of PCR systems to allow for many reviewers and passive readers but only text-based conversations along source code. In order to understand why teams would then have different ideas of the techniques, we adopted a view based on the concept of functional affordances which is essentially a critical realist account of such phenomena (Markus & Silver, 2008). It posits that technologies with real properties afford different functions to different actors in pursuit of their goals, depending on the actors’ embeddedness in generative structures (Markus & Silver, 2008; Leonardi, 2011). Transferred to peer-based quality assurance, this view suggests that the applied technologies have causal powers in providing the necessary conditions for conducting PCR with many members and PP with intense interactions. Both PP and PCR afford the correction of programming errors and enforcement of conventions; activities that necessitate only very limited interactions of few developers. Instead, when making use of the intense interaction afforded by PP, developers were enabled to give each other intensive process feedback regarding the task at hand and its potential completion. In turn, by including a larger number of actors in a PCR process, teams were afforded to give authors diverse outcome feedback from multiple perspectives that allowed improving the source code initially proposed.

Functional affordances are relational in that they link technical objects (such as a PCR system or the shared computer in PP) with the user or user group of these objects (Markus & Silver, 2008). In the case of applying PP and PCR within ISD teams, our earlier discussion suggests that the forms of use that PP and PCR afford to ISD teams may depend also on the team’s TMS structure. Thus, we aimed at understanding how a team’s TMS structure interacts with the structural properties of PP and PCR in affording different forms of use to different teams. In order to reach this goal, we engaged in selecting and preparing four in-depth case studies of ISD teams. When we entered the teams, we kept close track not only of how the developers leveraged the teams’ TMS during PP and PCR, but also of the different afforded functions they made use of. However, as we examined the motivations for and ways of conducting PP and PCR in these four teams in detail, we found that the use patterns we elicited during our initial exploration did not cover all the different functions that members felt PP and PCR afforded them. While PCR was often used to integrate broad feedback on the source code, there were also events in which authors required feedback not from many but only from "the right” single colleague (targeted feedback). Moreover, the PCR system was also used by developers to signal their activity and their simultaneously increasing expertise in task areas. Colleagues could later refer to the PCR system when looking for someone knowledgeable in a certain area. In the same vain, PP was often used to develop solutions jointly based on intensive process feedback and interaction, but some PP sessions were also conducted to show to a less experienced team member how tasks were approached by an expert in the area, that is they aimed at knowledge sharing primarily. Finally, there were also situations in...
which team members stated that two specific colleagues had to conduct targeted pairing because they were the only ones that could address a task based on their prior knowledge. Figure 3 summarizes these refined functional affordances of PP and PCR.

<table>
<thead>
<tr>
<th></th>
<th>Pair Programming</th>
<th>Peer Code Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>interaction required</td>
<td>Share domain knowledge</td>
<td>Correct errors</td>
</tr>
<tr>
<td></td>
<td>• work on a standard task</td>
<td>• partner corrects programming mistakes</td>
</tr>
<tr>
<td></td>
<td>• partners exchange existing knowledge</td>
<td></td>
</tr>
<tr>
<td>problem solving</td>
<td>Joint problem solving</td>
<td>Enforce conventions</td>
</tr>
<tr>
<td></td>
<td>• two random members work on problem in familiar domain</td>
<td>• partner highlights violations of code conventions</td>
</tr>
<tr>
<td></td>
<td>Targeted pairing</td>
<td>Integrate targeted feedback</td>
</tr>
<tr>
<td></td>
<td>• new problem in unfamiliar area</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• experts in most adjacent areas pair</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• elaborate problem &amp; jointly develop solutions</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3. Functional Affordances in Case Studies**

**In-Depth Case Studies: The Teams**

The four teams we examined more closely differed not only in their application of PP and PCR but also in the TMS structures they made use of. COMPILE develops a custom language compiler as a central part of the case company’s most innovative product. The team’s internal customers, that is several dozens of teams that work on this product, demand frequent delivery of new or improved features of the compiler. At the same time, errors and bugs on such a low level of the product would compromise the work on higher levels as well. Consequently, COMPILE not only has to develop requested features rapidly, but also needs to ensure high quality of work results. COMPILE’s members rely heavily on PCR to do so and have nearly all their code reviewed by at least one other team member. Moreover, there are different components of the compiler, and with those different technological specifics, of which only few of the developers have profound knowledge. In the past, the team has created expertise maps in order to show who had special knowledge in which areas. Today, team members are quite confident they know who in the team could give them helpful advice in any question regarding components of the compiler or technologies used. Even though the single developers have complete discretion which work package they want to work on, the team’s product owner could give us a priori quite a good picture of who was most likely to work on which tasks during a two weeks work iteration that we partially observed. In sum, COMPILE is a team with a highly sophisticated TMS structure with single developers possessing expertise in specific areas and their colleagues knowing these areas and making use of their expertise.

BPM develops one of the case company’s established core products for business process management and is not only responsible for developing a new software piece on top of the core product but also for maintaining and fixing issues with the underlying software. In the past, there were severe quality issues with this core product that resulted in today’s strong focus on quality in the mindset of both BPM and its management. To assure high quality, BPM relies on PP as well as PCR to a large extent. New code and bug fixes are usually developed in pairs and often additionally reviewed by peers. The team members are convinced that one central point that led to quality issues in the past was membership change with single experts of software
components leaving the team or even the company. As a consequence, BPM’s members now try to create a homogeneous distribution of knowledge. BPM has an established, team-wide vocabulary for referring to specific areas of the software, technologies, and issues. Only few areas remain where single experts hold larger amounts of knowledge singularly, but this is well known to the rest of the team. In sum, BPM has a moderately sophisticated TMS structure with vast shared encodings, only moderate specialization but large meta-knowledge about members, source code and the specialist areas.

DELIVER is the result of merging two teams several months ago and is responsible for all tasks previously residing with the old teams. The team provides software development infrastructure to other teams in the company to host and maintain their development projects in different system landscapes. DELIVER’s members develop most of their source code in pairs and often conduct PCR. Both, our observations and interviews made clear that tasks and specialized knowledge still resided with the same developers that had been dealing with them before the team merger. While members of the old sub-teams knew very well how expertise regarding technologies and service components they used to provide were distributed among former team members, this was much less so across the two sub-teams in DELIVER. For instance, when sub-team members paired, we observed frequent implicit and rapid agreement on how classes and methods should be named and what was deemed good architectural design. By contrast, across sub-team PP sessions involved much more discussions, arguments, and need for explicit alignment, also because single members often were not aware of non-functional requirements or underlying technologies in areas that had not been part of their old teams. In sum, DELIVER has a moderately sophisticated TMS structure on a team-level while the TMS structures of the two sub-teams are highly sophisticated.

SELL develops an online sales platform that is based on popular underlying customer relationship and enterprise resource management software. While maintaining and fixing also prior releases, the team develops advanced versions of the platform. To the largest part, this requires user interface development based on the provided functionality of the underlying software. Working hours of some team members have only little overlap, as several developers are employed part-time in this team. This might be one reason why PP, although conducted for most bug-fixes and parts of new feature development, is often done within the same pairs of developers with only little rotation. Compared to all other teams, SELL’s members had the most trouble communicating and coordinating; we observed this especially during PP when pairs struggled in agreeing on procedures or coming to common understandings of what an appropriate solution of single requirements should cover. This was exemplified by a PP session in which the pair had an easy task, namely the implementation of a binary flag to activate or deactivate some provided functionality. Confronted with a minor issue, the developers could for a long time not understand how their peer wanted to solve it (using an external editor vs. reconfiguring the programming environment). Even after settling this, the pair had major problems with understanding each other’s ideas, approaches, and proposed solutions. Interviews with SELL’s members and Scrum master also showed that not very much was known about others’ areas of expertise except for one acknowledged expert in the area of performance optimization. Nevertheless, it was perceived that members were starting to increasingly build up expertise in different areas, although single tasks resided only for a short time with the same members. In sum, SELL’s TMS structure is of low sophistication compared to the three other teams we examined in our study.

**Interaction of Functional Affordances and TMS Structure**

The results from our case study suggest that affordances of peer-based quality assurance techniques are not only influenced by the structural properties of the techniques, but also by the knowledge distribution within a team. In particular, we found that the techniques afford an increasing number of useful applications to ISD teams the more sophisticated their TMS structure is (i.e., the more specialized their distributed knowledge and the more commonly shared their meta-knowledge). While correcting programming errors, for example, would be afforded to any combination of developers by both techniques independent of any specialized knowledge or shared meta-knowledge, enforcing conventions would at least require common knowledge about code conventions in the technical domain they work in.

By contrast, PCR affords authors a way to signal activity in which area they are currently working if (i) the team members share the meta-knowledge about labels and language (i.e., encoding) in the technical as well

---

**General IS Topics**

components leaving the team or even the company. As a consequence, BPM’s members now try to create a homogeneous distribution of knowledge. BPM has an established, team-wide vocabulary for referring to specific areas of the software, technologies, and issues. Only few areas remain where single experts hold larger amounts of knowledge singularly, but this is well known to the rest of the team. In sum, BPM has a moderately sophisticated TMS structure with vast shared encodings, only moderate specialization but large meta-knowledge about members, source code and the specialist areas.

DELIVER is the result of merging two teams several months ago and is responsible for all tasks previously residing with the old teams. The team provides software development infrastructure to other teams in the company to host and maintain their development projects in different system landscapes. DELIVER’s members develop most of their source code in pairs and often conduct PCR. Both, our observations and interviews made clear that tasks and specialized knowledge still resided with the same developers that had been dealing with them before the team merger. While members of the old sub-teams knew very well how expertise regarding technologies and service components they used to provide were distributed among former team members, this was much less so across the two sub-teams in DELIVER. For instance, when sub-team members paired, we observed frequent implicit and rapid agreement on how classes and methods should be named and what was deemed good architectural design. By contrast, across sub-team PP sessions involved much more discussions, arguments, and need for explicit alignment, also because single members often were not aware of non-functional requirements or underlying technologies in areas that had not been part of their old teams. In sum, DELIVER has a moderately sophisticated TMS structure on a team-level while the TMS structures of the two sub-teams are highly sophisticated.

SELL develops an online sales platform that is based on popular underlying customer relationship and enterprise resource management software. While maintaining and fixing also prior releases, the team develops advanced versions of the platform. To the largest part, this requires user interface development based on the provided functionality of the underlying software. Working hours of some team members have only little overlap, as several developers are employed part-time in this team. This might be one reason why PP, although conducted for most bug-fixes and parts of new feature development, is often done within the same pairs of developers with only little rotation. Compared to all other teams, SELL’s members had the most trouble communicating and coordinating; we observed this especially during PP when pairs struggled in agreeing on procedures or coming to common understandings of what an appropriate solution of single requirements should cover. This was exemplified by a PP session in which the pair had an easy task, namely the implementation of a binary flag to activate or deactivate some provided functionality. Confronted with a minor issue, the developers could for a long time not understand how their peer wanted to solve it (using an external editor vs. reconfiguring the programming environment). Even after settling this, the pair had major problems with understanding each other’s ideas, approaches, and proposed solutions. Interviews with SELL’s members and Scrum master also showed that not very much was known about others’ areas of expertise except for one acknowledged expert in the area of performance optimization. Nevertheless, it was perceived that members were starting to increasingly build up expertise in different areas, although single tasks resided only for a short time with the same members. In sum, SELL’s TMS structure is of low sophistication compared to the three other teams we examined in our study.

**Interaction of Functional Affordances and TMS Structure**

The results from our case study suggest that affordances of peer-based quality assurance techniques are not only influenced by the structural properties of the techniques, but also by the knowledge distribution within a team. In particular, we found that the techniques afford an increasing number of useful applications to ISD teams the more sophisticated their TMS structure is (i.e., the more specialized their distributed knowledge and the more commonly shared their meta-knowledge). While correcting programming errors, for example, would be afforded to any combination of developers by both techniques independent of any specialized knowledge or shared meta-knowledge, enforcing conventions would at least require common knowledge about code conventions in the technical domain they work in.

By contrast, PCR affords authors a way to signal activity in which area they are currently working if (i) the team members share the meta-knowledge about labels and language (i.e., encoding) in the technical as well
as the application domain the author is working in and (ii) the author gains special expertise in this area. Without such meta-knowledge, team members are unable to understand what the author is actually doing even though they might see the changes in source code.

Figure 4 shows the previously identified patterns of use as cumulative affordances for increasingly sophisticated TMS structures of ISD teams. Figure 4 also shows the additional components of the team’s TMS structure that is leveraged in order to engage in the described use patterns. For instance, when applying PP for joint problem solving, pairs leverage partially specialized knowledge in addition to the common encoding necessary to share domain knowledge. Figure 5 shows exemplary events where the four teams applied PP and PCR in one of the afforded ways and how these teams leveraged their TMS structure in order to do so.
Generally, ISD teams with sophisticated TMS structures of highly specialized knowledge and commonly shared meta-knowledge are afforded the entire range of activities by the techniques. For instance, developers from one of the four teams, DELIVER, made use of all affordances that PP provided to them. Leveraging their moderate TMS sophistication on a team level, they deliberately applied PP at different occasions to distribute knowledge (share domain knowledge) and to solve existing problems (joint problem solving). However, within their two sub-teams with highly-developed TMS structures, they relied on targeted pairing of experts from adjacent areas of expertise to develop unprecedented solutions for new problems. In the same way, the members of COMPILE, used PCR to integrate targeted feedback from well-known experts on single components if their knowledge was required, but they also relied on simple error correction and convention enforcement affordances of PCR if they deemed a task rather simple.
In sum, PCR and PP afford the same functions (correct errors & enforce conventions) to ISD teams that have only rudimentary TMS structures or do not make use of the techniques’ structural properties. With increasing sophistication of their TMS structure, the same techniques afford these teams additional, different functions that rely more on their technological flexibility to involve participants and to allow relational interactions (cf. Figure 4 & 5).

**Implications on TMS structure**

Making use of afforded possibilities by leveraging required aspects of their team’s TMS structure, developers consciously or subconsciously exert TMS processes, of which some have the potential to yield an emergent effect on the TMS structure, in turn. Table 3 depicts the TMS processes triggered by the use of functional affordances of PP and PCR with continued reference to the exemplary events provided in Figure 5.

As discussed earlier, only team-wide changes in specialized knowledge and associated meta-knowledge in-

### Table 3. Functional Affordances of PP & PCR with Consequences for TMS

<table>
<thead>
<tr>
<th>TMS Processes Stimulated</th>
<th>Exemplary Events from Figure 5 Revisited</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Joint Affordances</strong></td>
<td></td>
</tr>
<tr>
<td>– Error Correction</td>
<td>none</td>
</tr>
<tr>
<td>– Convention Enforcement</td>
<td>none</td>
</tr>
<tr>
<td><strong>PCR Affordances</strong></td>
<td></td>
</tr>
<tr>
<td>– Broad Feedback</td>
<td>retrieval &amp; integration of feedback from several sources; storage of new meta-knowledge &quot;who knows what&quot;</td>
</tr>
<tr>
<td>– Signal Activity</td>
<td>retrieval &amp; integration of feedback from selected experts; storage of meta-knowledge &quot;who does what&quot;</td>
</tr>
<tr>
<td>– Targeted Feedback</td>
<td>retrieval &amp; integration of feedback from selected experts; validation of meta-knowledge &quot;who knows what&quot;</td>
</tr>
<tr>
<td><strong>PP Affordances</strong></td>
<td></td>
</tr>
<tr>
<td>– Knowledge Sharing</td>
<td>retrieval &amp; integration of actors' expertise; encoding of integrated application domain knowledge; storage of integrated knowledge by both redundantly</td>
</tr>
<tr>
<td>– Joint Problem Solving</td>
<td>retrieval &amp; integration of actors' expertise; encoding of integrated application domain knowledge; storage of integrated knowledge by both redundantly</td>
</tr>
<tr>
<td>– Targeted Pairing</td>
<td>retrieval &amp; integration of actors' expertise; encoding of integrated application domain knowledge; validation of meta-knowledge &quot;who knows what&quot;; storage of integrated knowledge by both redundantly</td>
</tr>
</tbody>
</table>

Exemplary events continued from Figure 5
fluence a team’s TMS structure. Peer code review by tendency addresses and affects more team members at the same time, though with less relational intensity than PP (cf. Table 3). With a well-developed TMS structure in place, PCR affords integrating targeted feedback as a lean and efficient way of knowledge retrieval that simultaneously allows for validation of the TMS between single actors (Brandon & Hollingshead, 2004). Both signaling activity and integrating broad feedback provide means to address many team members within a single session. To developers who voluntarily demonstrate specialized expertise in a certain area as an author or reviewer, PCR affords these functions to spread meta-knowledge of who knows what effectively. This requires, however, that team members have a common encoding for such knowledge in place. If an ISD team makes use of these affordances and leverages the underlying technology’s potential to include many actors, PCR allows for improving a team’s TMS drastically by distributing meta-knowledge associated with existing specialized knowledge.

Through intensive social interaction, PP by tendency creates more shared meta-knowledge as well as redundant knowledge between the single individuals than PCR (cf. Table 3). While this allows a pair to gain common understanding and better access to each other’s expertise in the future, it has not necessarily any consequences for the ISD team as a whole as the pair works isolated from the rest of the group. The emergence of a team level effect depends on the interconnection of the pairing developers with their colleagues. For example, rotating the pairs from task to task can lead to a diffusion of knowledge and meta-knowledge from one pair to the group level. However, the sophistication of a TMS structure relies on members that specialize in knowledge areas but commonly share the associated meta-knowledge. As a consequence, redundant storage of knowledge as it is typical for PP (cf. Table 3) may actually harm the TMS on a team level: if knowledge is shared by a specialized developer who is deemed the responsible expert for an area in the team, redundantly storing this specialist knowledge would reduce specialization in the team and would make responsibility for this knowledge area ambiguous. On the other hand, it does not have any negative impacts on the TMS structure, if knowledge is shared that is (i) not specialist knowledge or (ii) that is specialist knowledge but not covered by meta-knowledge. In sum, the techniques afford not only different functions depending on the TMS of a team; once a team makes use of an affordance, this distinctively influences the distribution of knowledge and meta-knowledge, that is also the TMS. The techniques can, therefore, provide complementary effects to teams that already have a sophisticated TMS structure in place. In teams with a rudimentary TMS structure they can only serve as substitutes for error correction and convention enforcement.

Discussion and Conclusion

This study was motivated by the assertion that the interaction among individuals implanted in modern software development techniques leads to thinking and knowledge that extends beyond individual developers. Whereas extant literature generally acknowledges cognitive effects in collaborative ISD development, the question how these techniques interact with the socio-cognitive structures of ISD teams remained so far unanswered (Davern et al., 2012). The goal of this study was to help address this gap and improve our understanding of the interplay of teams’ transactive memory system and peer-based quality assurance techniques. Whereas previous work mostly studied these techniques in laboratory experiments or open-source communities, we aimed at obtaining rich insights from in-depth case studies of ISD teams within a global software development firm.

In order to answer our research question, we integrated the concepts of functional affordances (Markus & Silver, 2008) and transactive memory systems (Wegner, 1987). The analysis of our four case studies yielded two major findings. First, we found that even though trained similarly, ISD teams within the case company applied the two techniques under consideration in varying ways. These patterns of use ranged from a mere four-eye principle with the goal of error correction to joint problem solving through intensive interaction in PP and soliciting targeted feedback through a PCR system. Second, the different patterns of using the techniques were theoretically linked to two distinct structures: the team’s TMS structure as well as the structural properties of the technology that underlies the techniques. As such, our findings clarify how TMS structure and technological properties interact and collectively enable teams to use the techniques in
specific ways: based on the TMS structure of a team, the techniques afford it different functions, but once one of these functions is put in use it distinctively influences the TMS in return.

By examining the relationship between transactive memory structures of teams and software development techniques that are applied at a dyadic or sub-group level this study makes a number of contributions to extant literature. As such, most of the existing literature has either focused on the socio-cognitive aspects of ISD teams without referring to the behavior that shapes and is shaped by these structures, or has studied software development techniques on an individual or dyadic level without considering effects on the thinking and knowledge that goes beyond individual cognition. As a recent exception, Sarker et al. (2011) studied the relation between individuals’ importance in the social network of ISD teams and knowledge sharing between individuals. As opposed to such a view that stresses the topology of a team, our study contributes to literature by emphasizing the nature and origin of the interplay between socio-cognitive structure and human behavior within ISD teams.

Moreover, this study contributes to previous work on peer-based quality assurance techniques. Most of the existing work has, on the one hand, studied only the direct effects of applying these techniques on code quality and, on the other hand, mostly examined the benefits of applying one technique as compared to solo programming. Only few studies have tried to contrast different techniques, such as PP and PCR, despite the fact that these particular two techniques are often seen as functional substitutes in terms of ensuring code quality (Müller, 2004, 2005; Rigby et al., 2012; Schmidt et al., 2012). By moving beyond direct implications on code quality, we are able to add substance to the debate if PCR may substitute PP within collocated ISD teams or if both techniques could be fruitfully combined.

These findings also have direct practical implications. Practitioners should be well aware of the current and aspired transactive memory system in their ISD team when they think about introducing PP or PCR. Our findings show that PP and PCR may indeed be functional substitutes if a team’s TMS structure is undeveloped or if the flexibility of the two techniques is not leveraged. In this case, the role of PP and PCR is indeed restricted to ensuring code quality. However, when accounting for the transactive memory structure of teams, we obtain a more differentiated picture. More specifically, PP and PCR may not substitute but complement each other if teams possess a well-developed transactive memory structure. Teams with a sophisticated TMS structure may benefit from PCR by drawing on existing specialist knowledge and by creating transparency about the development of new expert knowledge. When applying PP, developers are enabled to engage in close interaction and joint problem-solving, thus making expert knowledge more accessible through the development of more fine-grained meta-knowledge (e.g., on the credibility of the knowledge source or the application of the expert knowledge in the development process). In order to share this meta-knowledge with all team members, teams may again rely on PCR. Another way to distribute meta-knowledge across the team may be to ensure rotating pairs. This, however, potentially makes targeted pairing less feasible. Thus, for teams that heavily rely on targeted pairing and therefore rotate pairs to a lesser extent, PCR may complement PP as a tool to ensure the sharing of meta-knowledge.

Moreover, based on our transactive memory perspective, PCR may be particularly useful for new teams that often have a high degree of specialization, but have not yet developed the associated meta-knowledge. In the initial phase, team members may retrieve expert knowledge through integrating broad feedback. In fact, this is a situation that closely resembles open-source software communities, where PCR was originally established. Thus, PCR may be a valuable addition to PP particularly for newly established teams.

The findings of this study may lay the groundwork for future research on the interplay of ISD techniques and socio-cognitive structures. For instance, our findings indicate that affordances of peer-based quality assurance techniques have the potential to change a team’s TMS structure. Prior research showed that such impacts on groups are particularly strong when they make stable use of the same affordances over time (Leonardi, 2012). However, this requires individual agency and is prone to intended and unintended variation (Howard-Grenville, 2005). Further work could therefore take a dynamic perspective and examine under which conditions developers in ISD teams converge in their use of specific affordances of peer-based quality assurance techniques.
References


