The Transactive Processes of Social Coding: How Code Review Substitutes for Transactive Memory in Software Development Teams

Completed Research Paper

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

Thomas Kude
ESSEC Business School
Cergy Pontoise, France
kude@essec.edu

Christoph T. Schmidt
The Boston Consulting Group
Munich, Germany
schmidt.christoph@bcg.com

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

Abstract

Social coding refers to a modern and technology-mediated way of peer code review between software developers. Whereas social coding has a long tradition in open source communities, it is only recently becoming popular in co-located, agile information systems development teams. Although social coding constitutes an important mechanisms of quality assurance and organizational learning in open source, little is known about its effects on co-located and ongoing professional software development teams. We close this gap by examining the question of how and why social coding affects professional development teams' work outcome and transactive memory. Results from our questionnaire-based survey with 61 development teams of more than 300 developers, team leads, and product managers, suggest that social coding not only improves teams' software quality directly but can also partially substitute for their transactive memory system. This has implications for theory on software development and transactive memory.

Keywords: Agile software development, Information systems development, Transactive memory, Social coding, Team cognition, Code review

Introduction

Social coding is an increasingly popular, technology-based software development technique that brings together multiple software developers to work asynchronously on the same piece of source code via a code review system (Dabbish et al., 2012). In more detail, social coding refers to a modern way of peer code review in which a software developer authors a functional piece of software and uploads it to a code review platform. Fellow developers can then act as reviewers to examine, criticize and discuss it. The author revises the piece of software, potentially during multiple rounds of reviews and revisions, until it is ultimately rejected or accepted for addition to the shared code base of all developers (Beller et al., 2014, Pangsakulyanont et al., 2014, Rigby and Storey, 2011). Social coding has a long-standing tradition in open source development where it not only constitutes a central quality assurance mechanism but also serves for organizational learning (Lee and Cole, 2003, McIntosh et al., 2014). Currently, social coding is being increasingly adopted by professional software developers in for-profit companies. It appears especially

interesting that also co-located information systems development (ISD) teams of professional developers rely on this technology-mediated review practice (Rigby et al., 2012) although their settings are very different from the globally distributed, loosely coupled, and heterogeneous developer networks of open source communities from which social coding originates (Daniel et al., 2012).

Because prior research has predominantly been confined to the specific settings of open source communities, little is known about the effects of social coding in professional software development teams. What is more, it is currently unclear whether the necessary efforts for social coding translate to benefits in professional software development teams that work together over longer periods of time. Moreover, there have been assertions that social coding helps developers exchange knowledge (Bacchelli and Bird, 2013, Rigby et al., 2012), but research has yet to examine these assertions theoretically and empirically. We address this gap by focusing on software development teams' transactive memory systems, i.e., the socio-cognitive mechanisms that allow work groups to integrate and use the specialized expertise of their members. Transactive memory systems were found to be strong predictors of team performance in software development teams (Faraj and Sproull, 2000, Lewis, 2003, Lin et al., 2011, Oshri et al., 2008). Against this backdrop, we aim to answer the question:

How and why does the application of social coding affect the transactive memory of professional, co-located ISD teams and the software they develop?

In doing so, this paper has two objectives: (i) to develop a research model that explains how and why the use of social coding affects development teams' transactive memory systems and work results; (ii) empirically validate the proposed model in a field study with co-located, professional software development teams.

The remainder of this paper is structured as follows. We first review literature on social coding and transactive memory systems. We then develop our research model based on transactive memory theory and hypothesize effects of social coding on software quality and transactive memory system effectiveness. We then describe our research design and the specifics of our field study with 61 professional, co-located software development teams. Subsequently, we present the results of our study and discuss them in the light of prior research and theory on software development and transactive memory systems.

Conceptual Background

Social Coding

Social coding is a peer code review technique that is regularly carried out during daily routine by two or more software developers from the same work group when they work on the same piece of source code in order to achieve a commonly acceptable work result. Social coding thereby constitutes an informal, collaboration-based mechanism of ongoing quality assurance. Open source projects traditionally rely on social coding as a remedy to the absence of formal and hierarchical quality assurance mechanisms in their communities. They also use these ongoing, interpersonal but technology-mediated processes of peer code review as sources for organizational learning (Lee and Cole, 2003, McIntosh et al., 2014). Similar to academic peer review (Beller et al., 2014), social coding is often conducted differently in different communities, but always conforms to an elementary sequence of activities (Rigby and Bird, 2013): when the author of a piece of source code has finished programming, s/he submits it to a dedicated, group-wide peer code review system for feedback. Fellow developers volunteer to review this code change noting their critique as written text alongside the code and give the author the chance to revise and resubmit it if necessary. One or more rounds of review and revision are then concluded by the reviewer(s) either accepting the code and including it in their shared code base or rejecting it entirely. As different reviewers may not necessarily have the same opinion on each point of critique, open source communities typically rely on few, prominent decision makers in case of conflict, so-called "core committers" (Bacchelli and Bird, 2013). Social coding in small teams of companies, by contrast, is typically voluntary and usually targeted toward shared understanding, including collectively accepting minor shortcomings and rejecting major issues, than resolving all task conflict by means of hierarchy (Bacchelli and Bird, 2013).

Social coding (sometimes also called lightweight and modern code review (Beller et al., 2014, Pangsakulyanont et al., 2014, Rigby and Storey, 2011)) is less formal than traditional code inspections (Fagan, 1986) and does not require larger physical meetings. A number of experimental studies have

evaluated such traditional code inspections and their potential to identify defects through one or more reviewers (Müller, 2004, Müller, 2005, Phongpaibul and Boehm, 2006, Sauer et al., 2000, Tomayko, 2002, Winkler and Biffl, 2006). Findings from this stream of literature suggest that ISD teams do not find more defects during formal, meeting-based source code inspections with several team members than single pairs of developers do (Sauer et al., 2000, Winkler and Biffl, 2006). Social coding differs from these formal code inspections in several aspects: social coding is technology-mediated through dedicated peer code review systems and asynchronous, therefore there are no physical meetings of developers taking up time and resources; instead of examining large amounts of code during occasional formal inspections, social coding is conducted on small portions of code as part of ISD teams' daily work routine; instead of predefined inspection roles (Fagan, 1986), social coding relies on social connections between developers who volunteer for mutual reviews if they perceive their expertise to be adequate (Wang and Carroll, 2011). Social coding can consequently be argued to affect much more the social and socio-cognitive relationships between team members (Bacchelli and Bird, 2013) than traditional inspections, which primarily focused on defect detection (Rigby et al., 2012, Sauer et al., 2000). In fact, some scholars have argued that social coding may even constitute a less costly and functionally similar alternative to so-called pair programming - the most widely adopted agile development technique in ISD teams (Rigby et al., 2012).

Nonetheless, social coding has received extremely little academic attention (Beller et al., 2014, Bosu and Carver, 2014, Rigby, 2011, Rigby and Bird, 2013, Thongtanunam et al., 2015) although it is widely applied in software developing companies including Google, SAP, Erickson, Sony, Facebook, Microsoft, and AMD (Bacchelli and Bird, 2013, Rigby and Bird, 2013) and is associated with mentionable costs and efforts for code review systems, review activities, and revisions of all source code. Extant work is largely restricted to open-source settings with globally distributed and loosely connected individuals (Beller et al., 2014, Liang and Mizuno, 2011, McIntosh et al., 2014, Rigby et al., 2008, Rigby and Storey, 2011, Sutherland and Venolia, 2009) as opposed to ongoing ISD teams. Consequently, extant studies are often concerned with aspects of social coding that are idiosyncratic to the open source context such as gaining reputation within a community (Bosu and Carver, 2014), the social structure of large developer networks (Yang, 2014), and managing the sheer amount of review comments in large open source communities (Pangsakulyanont et al., 2014, Thongtanunam et al., 2015, Thongtanunam et al., 2014). Research suggests that social coding may be particularly effective in creating transparency in a group which can foster improved collaboration (Dabbish et al., 2012). In particular, Dabbish et al. (2012) suggest that developers not only produce better source code if it has to undergo a peer review but also observe and create opinions on fellow developers during their participation in social coding. Prior work in for-profit organizations moreover emphasizes the need to study socio-technical effects of social coding on a team level because it may affect ISD team members' possibilities to access one another's knowledge and learn about others' current activities (Bacchelli and Bird, 2013). In fact, extending one's personal skills and knowledge seems to be a primary motivation for individual developers' engagement in social coding (Bacchelli and Bird, 2013, Rigby et al., 2012, Rigby and Bird, 2013, Sutherland and Venolia, 2009). However, research has yet to examine the socio-cognitive consequences of social coding for ISD teams. In particular, it is questionable whether team members who work in the same office may benefit from any improvements in knowledge exchange that have been proclaimed for the distributed settings of open source communities.

Theory of Transactive Memory Systems

Davern et al. (2012) suggest that software development techniques may be closely related to ISD teams' socio-cognitive capabilities and call for research on this topic. We pick up on this idea and examine a type of social cognition that has proven particularly valuable for ISD teams as well as other work groups (Edmondson et al., 2007, Hollingshead et al., 2011, Ilgen et al., 2005, Lewis and Herndon, 2011, Peltokorpi, 2008, Ren and Argote, 2011): their transactive memory system. Through transactive memory systems, groups of people distribute information to and retrieve information from their members to coordinate and use individuals' expertise across the entire group (Wegner, 1987, Wegner et al., 1985). In particular, transactive memory systems allow work groups to engage in division of cognitive work, reduce cognitive load on individuals, and thus exceed the performance of groups that are not able to draw on their members' specific expertise (Hollingshead, 1998, Wegner, 1987). ISD teams with well-developed transactive memory systems show higher team performance regarding effectiveness, efficiency, knowledge integration, and the ability to transfer knowledge (Espinosa et al., 2007, Faraj and Sproull, 2000, Kotlarsky and Oshri, 2005, Lin et al., 2011, Maruping et al., 2009, Oshri et al., 2008). Transactive memory theory emphasizes the

specialization of single team members in different expertise areas that are integrated for team work by means of transactive processes. In more detail, it holds that work groups show higher team performance if they develop both (i) a transactive memory structure with specialized expertise, shared language, and shared meta-knowledge of who knows what and (ii) transactive processes for encoding, storing, and retrieving information (Lewis et al., 2007, Ren and Argote, 2011). Transactive processes represent human activities to assign shared labels to pieces of individuals' expertise (encoding), store knowledge with the appropriate team member (storage), and obtain task-relevant knowledge from team members based on their areas of expertise (retrieval) (Lewis et al., 2007, Ren and Argote, 2011, Rulke and Rau, 2000). Transactive processes are therefore the actions taken by team members to create, update, and leverage the transactive memory structure which is present in the memories of individual team members (Lewis and Herndon, 2011). Transactive memory structure, in turn, consists of actual task knowledge and meta-knowledge that is required to communicate about tasks and expertise in order to store, locate, and retrieve information (Wegner, 1987).

Despite Davern et al.'s (2012) call for research on team cognition and the use of ISD techniques, there is only scarce research on the role of ISD techniques in relation to transactive memory systems. Maruping et al. (2009) show that the performance impact of transactive memory systems in ISD teams can vary depending on team norms, such as shared code ownership and coding standards. Majchrzak et al. (2013) study individuals' knowledge contributions to wikis, frequently drawing parallels to ISD techniques such as refactoring. Their findings suggest that transactive memory systems help individuals relate their own expertise to their colleagues' and helps them decide how to use their own knowledge for collaboration (Majchrzak et al., 2013). These results indicate that transactive memory systems possibly also influence how ISD teams are able to use ISD techniques. But whether and how this is the case remains unclear.

Research Model and Theory Development

Software Quality

This study focuses on the effects of social coding in ongoing, professional software development teams. Such ISD teams work on the same product over long periods of time and face challenges and goals that differ from the ones of teams working in short-lived projects or open source communities (Austin, 2003, Choi et al., 2010). As such, the fixed time line of ISD projects typically leads to an evaluation of software development teams not only based on the results of their work but also based on their adherence to the project's time line and its predefined resources, i.e., their process efficiency (Faraj and Sproull, 2000, Guinan et al., 1998, Nidumolu, 1995). By contrast, software development teams that develop, maintain, and extend a software product, potentially over several years, are much less bound to the short-framed time restrictions of a typical ISD project. By consequence, it becomes more desirable for such ongoing teams to create a product that can be maintained efficiently, effectively satisfies customer demands, and can easily be extended in the future. Software quality thereby becomes the center of attention. In combination with social coding's history in quality assurance, this led us to consider software quality as the central dependent and economically relevant variable of this study.

Software quality does not exclusively refer to the absence of programming errors but also to the adequacy of a software product for internal and external software stakeholders (Sommerville, 2012, p. 712). Internal software quality refers to properties of a software product's source code that facilitate its maintenance over time and flexibility for change (Mc Connell, 2004, p.483). This includes that source code be written, formatted, and documented in a way that allows for grasping its purpose and implementation specifics, thereby facilitating the identification of hidden flaws, changes, and extensions in the future. In addition, high-quality software meets external customer demands in that it is responsive to user needs and is efficient in its operation (Liang et al., 2010).

Several prior studies have shown that ISD teams rely on the effective combination of differentiated expertise through their transactive memory system to create software that is of high quality regarding both internal and external dimensions (Faraj and Sproull, 2000, Lin et al., 2011, Oshri et al., 2008). In particular, they rely on their team members developing specialized expertise that can be used by the entire team by means of established transactive processes (Lewis et al., 2007). A group of diverse experts may leverage advantages of specialization during knowledge-intensive software development and may more easily come up with creative solutions that match customer needs if they have appropriate transactive processes in place to

retrieve this expertise (Fægri et al., 2010, Gupta and Hollingshead, 2010). Moreover, single team members are known to perform better and make less mistakes in their own area of expertise if they believe that other team members do not have the same, redundant knowledge and cannot make up for any errors they commit (Hollingshead, 2000). We therefore hypothesize:

H1: The more developed a team's transactive memory system, the better the software quality provided by the team.

Social Coding in Co-located ISD Teams

During social coding, multiple developers examine the same piece of source code from different perspectives and at different points in time (Dabbish et al., 2012, Rigby et al., 2012). While the author of a piece of software may be trapped in his or her line of thinking during development, reviewers only see the resulting piece of source code and evaluate it for its quality without being trapped in the same reasoning as the author. ISD teams may thereby more readily be able to create software that complies with external customer needs, is structured and formatted in an intuitive way for easy maintenance, and is generally free of programming errors (Bacchelli and Bird, 2013). The simple fact that four eyes examine the same piece of code may consequently result in quality improvements already. In line with this reasoning, prior studies have found clear quality impacts of social coding in open source communities and have suggested that these benefits may transfer to teams of professional software developers (Lee and Cole, 2003, Rigby and Bird, 2013).

Importantly, however, previous studies also suggest that developers use social coding as a means of knowledge exchange (Bacchelli and Bird, 2013) and for inferences about their own role compared to group members' (Dabbish et al., 2012). In transactive memory research, technology has primarily been studied as a facilitator for transactive memory system development in work groups as well as in larger organizational units (Nevo and Ophir, 2012, Ren and Argote, 2011). Knowledge management systems and expert directories can provide individuals with explicit meta-knowledge about who knows what and support team members in learning about the expertise of their co-workers which leads to encoded meta-knowledge in their transactive memory structure (Choi et al., 2010, Nevo and Wand, 2005, Peltokorpi, 2004, Yuan et al., 2007). Social media can additionally provide social cues to facilitate decisions whom to supply with information and from whom to request information (Nevo et al., 2012). This entire stream of research implicitly holds that information systems and communication technologies complement the transactive processes conducted between individuals and help create, update, and sustain effective transactive memory structures (Oshri et al., 2008, Ryan and O'Connor, 2013)). Studies that find this complementarity typically examine groups that are challenged in transactive memory system development, for example by global distribution (Kotlarsky and Oshri, 2005, Lewis, 2004, Oshri et al., 2008, Su, 2012), a lack of tacit knowledge (Nevo and Wand, 2005), or short-lived projects (Lewis, 2004, Yuan et al., 2007). Co-located ISD teams, however, do not face such challenges and can freely develop their transactive memory by learning about one another and their tasks during face-to-face interactions. In such settings, technology-based interactions during social coding may not have any effect in fostering transactive memory but may instead provide an alternative way to execute transactive processes.

In line with such thoughts, Lewis and Herndon (2011) argue that technology may act as a substitute for transactive memory systems if it adequately emulates transactive processes. Others have found that teams sometimes rather rely on information systems to keep track of specialized expertise than to build up a transactive memory structure (Gray, 2000) as the latter may incur higher coordination costs (Griffith and Sawyer, 2010). However, there are rarely measureable benefits of information systems such as expert directories that primarily explicate who holds knowledge in an area (Child and Shumate, 2007, Nevo and Ophir, 2012) because expertise location alone is useless until expertise is actually retrieved and applied to a team task (Choi et al., 2010, Lewis and Herndon, 2011). By contrast, social coding as a technique, based on the underlying platform for peer code review, may actually allow for exactly executing all relevant transactive processes: authors in teams that rely on social coding may gain new knowledge while working on a development task and discussing their solution with reviewers. Other team members' expertise may be located in several ways during social coding: by searching old reviews and revisions in the system, by identifying relevant parts of existing code in the system together with the team members who last worked on it, or by simply broadcasting requests for review to all team members who then decide whether their expertise is worth contributing to the review. Having retrieved information and knowledge in the form of

review comments, an author directly applies it to create a revised version of the code change at hand. Finally, a revised and accepted piece of code which is understandable to the author as well as the reviewers is stored traceably together with data about its author, its reviewers, and the review comments. Future authors may therefore more easily understand this piece of source code, find team members experienced in this code, and even rely on old review comments irrespective of the authors' meta-knowledge of who knows what in the team.

Intensive and team-wide social coding may thereby decouple these transactive processes from the actual body of knowledge and meta-knowledge held by the team members. Social coding may consequently partially substitutes transactive memory structure in its function of coordinating expertise in ISD teams. According to Lewis and Herndon (2011), a real substitution of transactive memory should result in both a direct positive outcome effect of the substitute and an attenuation of the positive effect of a team's transactive memory. Consequently, we complete our proposed research model (Figure 1) by hypothesizing:

H2: The more intensively a team conducts social coding, the better the software quality provided by the team.

H3: The intensity with which a team applies social coding negatively moderates the positive effect of its transactive memory on software quality such that the positive effect of transactive memory on software quality is weaker for teams with a high intensity of social coding.

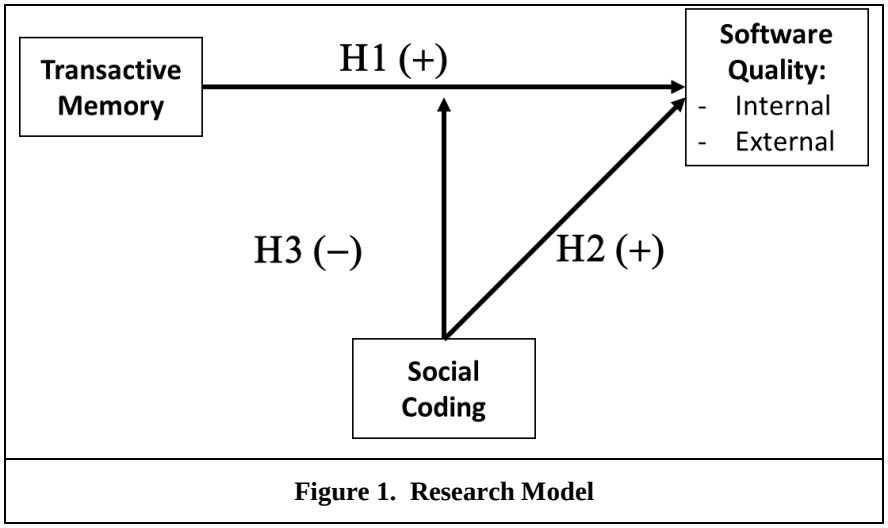

Figure 1. Research Model

Research Design

In order to answer our team-level research question, we conducted a questionnaire-based survey of software development teams of a large multinational software company. The selected case company is one of the world's largest enterprise software companies with more than 75,000 employees worldwide, locations in more than 130 countries, and a yearly revenue of over EUR 20 billion in 2015. It provides a broad range of enterprise software products, such as enterprise resource planning, supply chain management, customer relationship management, and business intelligence, to large, mid-sized, and small client enterprises. The chosen company is particularly interesting for research on social coding as its ISD teams work based on the agile ISD method Scrum (Larman and Vodde, 2010, Schwaber and Beedle, 2002) and are free to choose whether and how to apply specific ISD techniques. Consequently, teams and individual members may tend to reflect more on their application of practices (Janz and Prasarnphanich, 2009) and individual opinions may be of high value during the investigations. Most of the company's teams furthermore received centralized trainings in agile ISD techniques, including social coding, within the scope of a recent organizational transformation toward agile software development. In addition, it is the company's policy to minimize the number of virtual collaborations within single teams. That is, ISD teams of this company are typically co-located in the same office space.

Survey Design and Method

In order to fully appreciate the complementary perspectives of team members, team leads, and product managers, we created three different questionnaires so that these actors could be queried about aspects they could arguably answer best. This design increases accuracy of the collected information and at the same time reduces common method bias (Podsakoff et al., 2003). We asked product managers to describe and rate characteristics of the software product that was developed. In particular, product managers assessed the quality of software developed by their corresponding teams. Product managers are arguably better sources for assessments of software quality than team members as they can compare different components of a product and see the resulting software more holistically in the scope of a final product. Team leads were asked for details about descriptive characteristics of their team, existing team-wide norms, and the overall development process. In addition, they provided supplemental details on the use of social coding in their team such as the type of code review system applied. Team leads could thereby provide information that did not need to be gathered redundantly per team member. Finally, individual team members provided information on the use of social coding and the transactive memory system of their teams. This is in line with the idea that team members may arguably give more accurate information on their immediate actions and interactions than any third party. Such an assessment is therefore appropriate for constructs like a group's transactive memory that emerge from actions and interactions of single team members (Kozlowski and Klein, 2000).

Sample and Data Collection

As most of the teams in the studied company work according to the project management method Scrum, the product manager and team lead roles of the questionnaires were completed by roles specific to a Scrum context: "product owners" and "area product owners" completed the product manager questionnaires for teams that worked on their product and about the software they developed. "Scrum masters" completed the team lead questionnaires for their team. This role is compatible with the idea of a team lead as it is a Scrum master's defined role responsibility to ensure smooth collaboration within the team and to shield it from external forces that may interfere with team goals (Schwaber and Beedle, 2002). Finally, single software developers answered the team member questionnaires.

Questionnaires were printed on paper and filled in by all respondents: product managers received and returned the questionnaires by mail, team leads and members answered the questionnaires individually during physical team meetings with at least one of the researchers. In total, we received complete data for the variables in our model, including the ones collected from team leads and product managers, from 61 teams. We excluded developers' responses with missing data and extreme outliers in transactive memory items. In sum, we obtained a usable sample of 313 complete developer responses from 61 teams that corresponded with 61 complete ratings by team leads and product managers.

Average response rate for these teams was 69% of the team members. Average team size, including team lead and developers, was 8.9 persons and ranged between 3 and 13 persons (Table 1). The sample drawn from the case company represents a broad variety of teams in different contexts and an even broader variety of individual backgrounds.

Table 1. Sample Characteristics

Team Size	Less than 7 team members	11%	Project Size	Less than 3 teams in project	34%
	7 – 8 team members	28%		3 – 5 teams in project	38%
	9 – 10 team members	41%		6 – 8 teams in project	13%
	More than 10 team members	20%		More than 8 teams in project	15%

Variance-Based Model Estimation

We employed a variance-based approach to structural equation modeling with partial least squares (PLS) to estimate the measurement and structural models of this study (Chin, 1998, Kock and Lynn, 2012). We made this choice for two reasons: first, in contrast to other types of multivariate regressions, structural

equation modeling takes measurement error into account, which is particularly useful in studies with latent variables that must be measured indirectly such as transactive memory systems (Gefen et al., 2011). Whereas covariance-based approaches can control for measurement error, the employed variance-based approach can reduce its effects (Gefen et al., 2011). PLS was preferred over covariance-based structural equation modeling in this study because it preserves more information regarding the endogenous constructs than covariance-based structural equation modeling (Chin, 1998). Second, the chosen PLS estimator does not make distributional assumptions about the sample data which is particularly useful in case of smaller data sets (Kock, 2013).

To estimate the outer and inner models, we employed WarpPLS 5.0 with PLS regression, linear path estimation, and jackknifed resampling as recommended for smaller sample sizes (Kock, 2013). Importantly, the chosen measurement model estimator, namely *PLS regression*, does not allow for the structural model to influence the measurement model (Kock, 2013). This is different from the traditional PLS modes A, B, and M, which iterate between estimating measurement model and structural model with the goal to maximize the explained variance in the dependent variable (Chin, 1998). Whereas PLS modes A, B, and M have been criticized for a tendency toward overestimation (Rönkkö and Evermann, 2013), PLS regression is more conservative and has explicitly been excluded from such critiques of PLS (Goodhue et al., 2013). Consequently, significance levels and explained variance calculated with the chosen PLS regression algorithm can be expected to be lower but less inflated than with traditional PLS modes (Kock, 2013).

Measures

We collected data on social coding and transactive memory from team members and team leads and data on software quality from product managers. For our team-based study, we created team level averages for each item and used these team level scores to conduct our analyses.

Social Coding

Social coding practices interconnect the different members of software development teams. By consequence, it is reasonable to define the intensity of social coding use in a team not only based on the frequency with which single team members employ social coding practices but also based on the overall use of the technique as an interconnecting practice between team members. In line with this idea, we devised a measurement instrument that allowed us to capture not only individuals' use of social coding but also the use of social coding as an interconnecting practice. In our team level study, we asked each participating developer to reference three specific team members (A, B, and C) and to indicate his or her agreement with the statement that the other person and the developer frequently reviewed each other's code (see Table A.1). We chose a measure of frequency of mutual reviews because it is a direct, behavioral measure of social coding and interaction frequency has previously been related to socio-cognitive activity in groups (Majchrzak et al., 2013).

We tailored a procedure to maximize the coverage of such bidirectional relations for each team. By means of specially developed sign plates with numbers that identified each developer of a team, we randomly assigned each respondent his or her references A, B, and C. The sign plates with ID numbers were placed visibly for all team members, but only the respondents themselves could clearly see for which team members they were asked to complete the questionnaire. We designed the plates in a way that collected the highest possible number of relations between responding team members. Per team, we thereby collected a directed graph of '3 x the number of responding members' dyadic relations for the items of social coding.

In addition, team leads provided information on the use of code review systems so that the amount of peer code review conducted in a team could clearly be distinguished from simply looking at one another's code and from conducting centralized meeting-based code reviews as defined by Fagan (1986). We only considered developers' scores for peer code review greater than zero if the team leads reported that the team used a peer code review system and conducted peer code review primarily based on this system.

Transactive Memory System

In line with the majority of questionnaire-based studies on transactive memory systems (Nevo and Ophir, 2012, Ren and Argote, 2011), we measured teams' transactive memory based on the reflective measurement

scale developed by Lewis (2003) that contains items reflecting specialization, credibility, and coordination of knowledge in the team. This scale measures both the differentiation of transactive memory structure and the effectiveness of transactive processes at the same time (Lewis, 2003, Lewis and Herndon, 2011). As previous research has argued that measuring a group's transactive memory should be based on the perceptions of the group's members (Lewis, 2003), single developers assessed their team's transactive memory along the items that can be found in Table A.1 in the Appendix.

Software Quality

We measured overall software quality in two distinct but equally necessary dimensions: internal and external software quality. These dimensions of quality are in line with both prior ISD research and practice (Mc Connell, 2004, Salleh et al., 2011). We conducted a series of interviews with 15 project managers at the target company of our study to elicit the indicators of these two dimensions that were most appropriate in the company's context. We pretested the measures in a student sample and with selected project managers. In the end, we measured external quality as the satisfaction of customer needs and the delivery of required functionality by each team. Similarly, internal quality delivered by a team was based on ratings of the maintainability and readability of developed source code and on the adherence to company-internal task completion criteria (cf. Table A.1). Although research and practice have proposed also other criteria of software quality (Mc Connell, 2004, Salleh et al., 2011), the chosen measures appeared adequate as they arguably reflect the internal and external facets of software quality, were appropriately contextualized, and performed well in pretests.

We conceptualized overall software quality as a formative second-order construct based on the two reflectively measured first-order dimensions internal and external quality. We made sure that a formative specification of the latent second-order construct overall software quality is appropriate both from a theoretical and a statistical point of view (Bollen, 2002, Cenfetelli and Bassellier, 2009, Petter et al., 2007). As such, it can be argued that high overall quality theoretically *results* from two relatively independent dimensions: external quality in the sense that a software provides the necessary functionality that is important to the client, and internal quality in the sense that it is constructed in a way to allow that development and ongoing maintenance processes complete without major impediments caused by the software product. Our concept of software quality captures both these two dimensions that may vary relatively independent from each other but provide a more comprehensive meaning of software quality together. By consequence, a formative specification appears appropriate (Petter et al., 2007). According statistical post-hoc evaluations (Cenfetelli and Bassellier, 2009, Petter et al., 2007) were conducted (see below).

Controls

Like other prominent studies in this area, we controlled for team size (Maruping et al., 2009). This appears reasonable as group size can have effects on both socio-cognitive structures of a group as well as its performance (Lewis et al., 2007). Team leads provided this number.

Aggregation to Team Level

We made sure that statistical requirements for team level aggregations were fulfilled by each variable collected on the level of individual team members (Bliese, 2000). As such, we controlled for sufficient within-group agreement for transactive memory and social coding scales by calculating average $r_{wg(j)}$ scores, which were comfortably greater than 0.7. Moreover, ICC_1 scores above 0.3 and ICC_2 scores above 0.7 indicated acceptable effect sizes and reliability of averaged team level ratings respectively.

Measurement Model Validation

A confirmatory factor analysis was conducted for all first-order variables in order to establish convergent and discriminant validity (Hair et al., 2009). All indicators loaded significantly ($p<.01$) on the respective first-order constructs with loadings greater than 0.5 and minor cross-loadings below 0.5. Indicator cross-loadings were generally more than 0.2 lower than actual loadings which is commonly seen as sufficient to indicate discriminant validity (Hair et al., 2009). Nevertheless, we checked for multicollinearity by calculating block variance inflation factors and full collinearity inflation factors (Cenfetelli and Bassellier,

2009). Both ranged comfortably below the threshold of 2.5, thus indicating the absence of relevant lateral and vertical collinearity (Kock and Lynn, 2012).

For all first-order constructs, composite reliability was well above than 0.8, Cronbach's alpha were greater than 0.7, and the average variances extracted were above 0.5. Square roots of the average variances extracted were comfortably greater than any inter-construct correlations. In sum, reliability as well as convergent and discriminant validity could be established for all first-order constructs.

Regarding the formative second-order variable of software quality, we took a two-step approach. We first estimated composite scores for internal and external quality to then use these scores for further estimations of the structural model (Ringle et al., 2012). This approach is valid and recommended because the structural model cannot influence the measurement model if applying PLS regression as an outer model algorithm (Kock, 2013). The weights of the composite scores were highly significant on the second-order variable ($p<.01$) with standard errors well below 0.7 and variance inflation factors below 2.5. This indicates non-redundancy of dimensions of the formative construct (Cenfetelli and Bassellier, 2009) and ultimately construct validity as well as the absence of multicollinearity (Kock, 2013, Kock and Lynn, 2012, Petter et al., 2007).

Lastly, we tested for common method bias (Podsakoff et al., 2003) via a full collinearity test through the assessment of full collinearity variance inflation factors as proposed by Kock and Lynn (2012). For all constructs, full collinearity variance inflation factors were well below the conservative threshold of 2.5, indicating the absence of common method bias in a variance-based approach to structural equation modeling such as PLS (Kock and Lynn, 2012).

Structural Model Evaluation

Table 2 outlines indicators giving hints toward model fit in PLS - although there is no generally accepted overall goodness-of-fit measure comparable to covariance-based structural equation modeling (Gefen et al., 2011, Kock, 2013). In our case, Tenenhaus' goodness of fit measure ranks well above the threshold of 0.36 suggested by Wetzels et al. (2009), indicating the addition of each latent variable to the model adds predictive as well as explanatory quality instead of spurious effects on R^2. Average block variance inflation factors and average full collinearity variance inflation factors below 3.3 further show that neither vertical nor lateral collinearity are of concern to our results (Kock and Lynn, 2012).

Table 2. PLS model fit indicators

Indicator	Value	Acceptance threshold (Kock, 2013)
Tenenhaus GoF	0.421	Large fit if > 0.36
Average block VIF	1.242	acceptable if <= 5, ideally <= 3.3
Average full collinearity VIF	1.309	acceptable if <= 5, ideally <= 3.3

Results

Figure 2 depicts the resulting standardized path coefficients, significance levels, and explained variance for the hypothesized effects. Results lend strong support to our three hypotheses. As such, ISD teams' transactive memory systems are found to positively affect overall software quality (supporting H1). Moreover, social coding is found to have two significant effects: first, the team-wide application of social coding positively influences the overall outcome of ISD teams' work (supporting H2), namely the internal and external quality of the software they develop. Second, results also support H3, showing that social coding negatively moderates the effect of transactive memory on software quality. Team size does not have a significant effect on overall software quality.

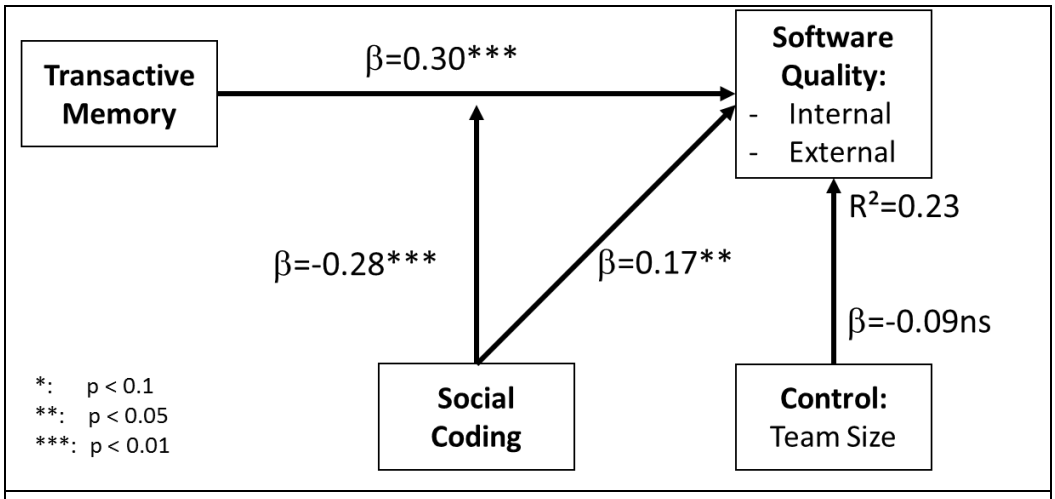

Figure 2. Research Model

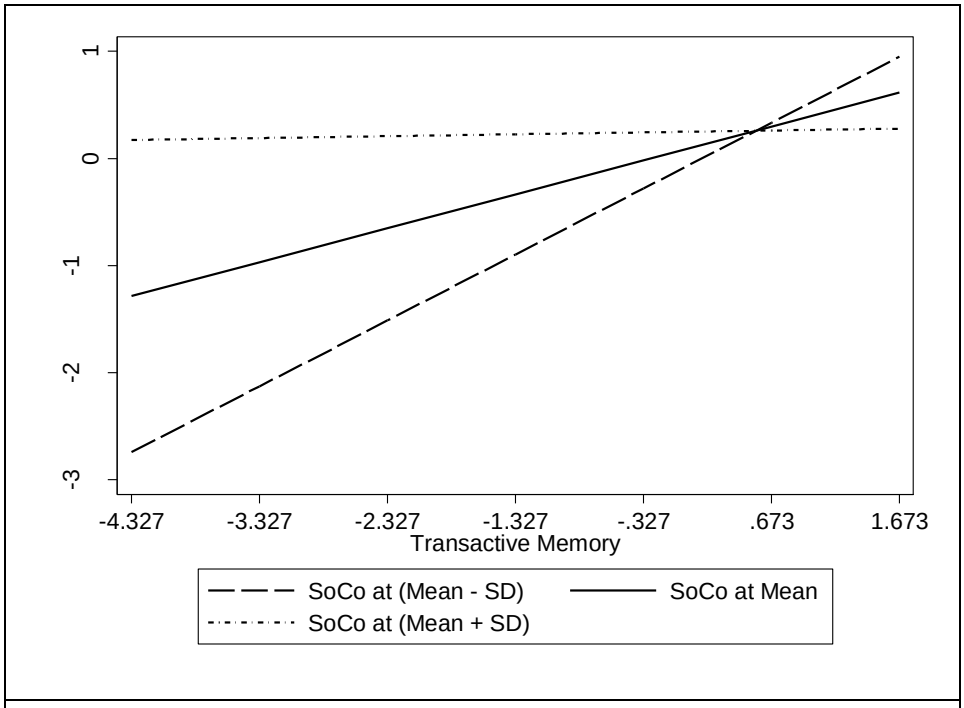

Figure 3. Interaction of TMS and Social Coding (standardized values)

In total, our model explains 23% of teams' software quality. R^2 values of this magnitude are quite common for studies on team cognition and performance in top tier IS journals. As such, Maruping et al. (2009) explain 30% of technical software quality, Faraj and Sproull (2000) explain 30% of team effectiveness, Choi et al. (2010) explain 24% of teams' performance, and Liang et al. (2010) explain 27% of software quality.

Figure 3 depicts the expected interaction effect of transactive memory and social coding at the sample mean of social coding plus/minus one standard deviation. Closer examination shows that transactive memory improvements are very effective on quality in settings with low intensity of social coding. In teams that conduct social coding intensively, however, higher levels of transactive memory are associated with marginal improvements in software quality only.

As we hypothesized, social coding has at the same time a significant positive effect on software quality and negatively moderates the effect of transactive memory on software quality. That is, the extensive application of social coding is associated with improved team outcomes in terms of software quality but reduced quality effects of transactive memory.

Discussion

Before discussing our study's contributions to literature on software development techniques and to transactive memory theory, we need to highlight several limitations of our study. First, we conducted our empirical investigation within the boundaries of one particular software development company. Although this setting provided a particularly informative context for studies on social coding techniques and helped us exclude effects of varying organizational cultures across teams, we highly recommend further studies investigate socio-cognitive effects of social coding in other development contexts and across companies. As our sample consisted of teams and individuals with quite heterogeneous backgrounds, we believe that our findings generalize to other development teams. Nonetheless, we call for more research validating and extending our findings in different team-based settings. For example, outsourcing projects with mixed teams of client and vendor employees may provide a very different and fruitful context to further scrutinize the effects of social coding techniques on the integration of differentiated team member expertise.

Implications for Theory

This study contributes to literature on software development and transactive memory theory in several ways. Prior studies on social coding, particularly on peer code review, were largely restricted to open-source communities and did not account for socio-cognitive effects. Our study is one of the first to demonstrate that technology-mediated social coding, which originated from loosely-coupled and globally distributed open-source communities, is beneficial even in settings of relatively small, professional software development teams working in the same room. It is not a waste of time but actually a driver of software quality that conforms to both external customer needs and internal maintenance requirements. Our findings thereby support and extend prior results from open source (Beller et al., 2014, McIntosh et al., 2014, Rigby, 2011) and organizational studies (Bacchelli and Bird, 2013, Rigby and Bird, 2013).

Scholars have recently highlighted that theoretically anchored research on software development is, at best, sparse (Mangalaraj et al., 2014). We add to this small but growing body of research by leveraging transactive memory theory to explain why social coding actually affects the software developed by teams. Our study empirically shows and theoretically explains that there is more than a mere direct effect of social coding on software quality that results from the simple fact that more developers scan each piece of source code for errors and deficiencies; social coding also helps developers exchange knowledge and contribute differentiated expertise to one another's development tasks. With possibilities to broadcast review invitations to all team members and to examine persistent data from old reviews and revisions, authors can quickly elicit relevant expertise in their team, retrieve review comments from their expert colleagues, and address their critique in a revised solution without detailed meta-knowledge of who knows what. Findings show that social coding can thereby partially compensate for deficiencies in a team's transactive memory structure. On the one hand, this distinguishes peer code review in organizational, co-located ISD teams from large open source communities where finding a reviewer with appropriate expertise often constitutes an overly complicated venture (Thongtanunam et al., 2015, Wang and Carroll, 2011). On the other hand, it also lends some support to scholars' assertions that knowledge exchange constitutes a pivotal element of social coding in for-profit settings and that social coding may increase the mutual awareness of developers (Bacchelli and Bird, 2013, Rigby and Bird, 2013). However, it also calls to attention that prior research has examined knowledge exchange only during single social coding sessions between individual developers. We are one of the first to acknowledge wider implications for entire ISD teams. Future research should therefore examine if social coding not only alters the effect of transactive memory on software quality but also other important team outcomes, such as creativity, which have been related to transactive memory in the past (Tiwana and Mclean, 2005).

Our study also has implications for transactive memory theory. First, our findings add to prior literature that emphasized the role of expertise integration in software development teams. In particular, prior work has shown that team norms, such as collective code ownership and coding standards, can strengthen or weaken the effects of expertise integration in software development teams (Maruping et al., 2009). We

extend this stream of research with results that suggest that technology-mediated social coding particularly has such a weakening effect because it not only allows teams to store knowledge and meta-knowledge but directly helps find and retrieve team members with adequate expertise on a piece of source code. Second, prior work has identified and developed technologies that aid in externalizing and capturing knowledge and meta-knowledge about team members (Kotlarsky et al., 2014, Nevo et al., 2012). However, Lewis and Herndon (2011) argue that current information systems fail to adequately substitute group level transactive memory systems because of their inability to emulate transactive memory processes. By consequence, technologies that allow for codification of transactive memory structure are typically not found to have substantial performance impacts (Lewis and Herndon, 2011). With social coding, we find a technology-based development technique that has both direct quality impacts and compensating effects on transactive memory effectiveness.

In more detail, our findings suggest that, rather than the technology itself, it is the team-wide and frequent application of social coding, entailing the use of social coding platforms to upload source code, solicit reviewers, and persistently discuss reviews and revisions, which actually creates such a substitutive effect. This suggests that a search for purely technological alternatives to transactive memory systems (Lewis and Herndon, 2011) may be doomed to failure. Socio-technical elements, including prescribed interactions with technology, may constitute the procedural factors of social coding that are required to actually substitute transactive memory systems. Future research may therefore aim to understand which generalizable properties afford the codification, effective maintenance, and use of this meta-knowledge. These results may then be a starting point for innovative design science research to create more effective and efficient technology-based techniques for expertise coordination in work groups in other contexts.

Implications for Practice

Lastly, our study also has a number of practical implications. First, managers of software development teams learn that the introduction of social coding indeed has the potential to significantly improve the quality of software developed by their teams, also in settings of co-located teams. Second, social coding may also help managers balance workload over multiple developers. Even if component experts may not have the time to implement a desired feature, they may serve as reviewers for proposed implementations. Management could consequently give feature implementation tasks to less experienced team members that could then acquire expert feedback on their solutions through social coding. This could move the firm-internal division of work more into the direction of contribution management in open source communities (Lee and Cole, 2003). Software development teams that discuss the introduction of social coding learn from our study that there are interaction effects of social coding and transactive memory: social coding is particularly useful for teams that do not possess a well-developed transactive memory system, for example because they have faced membership change or only recently started working as a team (Ellis et al., 2008, Lewis et al., 2007). The additional benefits of social coding, however, grow smaller with increasing quality of teams' transactive memory. Finally, our results imply that teams aiming to maximize the effects of social coding should engage all team members in frequent sessions of social coding and connect all individual developers with one another over time.

Conclusion

The goal of this study was to better understand the effects of social coding in professional, co-located ISD teams. In particular, we aimed to understand the socio-cognitive consequences of social coding. We leveraged transactive memory theory to explain the effects of social coding beyond direct quality impacts. We hypothesized that social coding allows development teams to conduct transactive processes for leveraging the specialized expertise of their members without requiring extensive transactive memory structures. Our survey study of 61 co-located software development teams lends support to this argument. More specifically, our results show that team-wide social coding not only drives software quality in terms of internal and external quality metrics; it also absorbs the positive outcome effect of ISD teams' transactive memory and can thereby be argued to substitute transactive memory structure. Social coding consequently constitutes much more than a simple mechanism for defect detection in ISD teams. Instead, it helps these teams to quickly leverage the specialized expertise of their single members without the need for extensive a-priori knowledge about one another. Social coding may therefore very well be an important, integrative buffer mechanism to keep team performance high even during times of disturbances that would otherwise

shake the foundations of an entire team's knowledge structure (Lewis et al., 2007). Whether and to what extent this powerful mechanism can be transferred from ISD teams to higher and even more impactful organizational levels, however, remains a promising question for future research.

Appendix: Measurement Items

Table A.1. Measurement Items

Variable	Items	Respondent (Scale)	Source
Social Coding	(You and colleague A:) We frequently review each other's code. (You and colleague B:) We frequently review each other's code. (You and colleague C:) We frequently review each other's code.	Developer (7 point Likert)	Developed
	If the team also does offline code reviews, how many reviews are not documented in the system at all?	Team Lead (% of reviews)	
	Does the team use a peer code review system? If so, which one?	Team Lead (Git Gerrit, Review Board, other)	
Transactive Memory System	Each member of my team has special expertise. Different team members are responsible for expertise in different areas. The expertise of several different team members is needed to complete our deliverables. I am comfortable accepting work suggestions from other team members. I am confident relying on the information that other team members bring into discussions. I trust that other members' task-related knowledge is credible. Our team works together in a well-coordinated fashion. We accomplish our tasks smoothly and efficiently. There is often confusion in our team about how we will accomplish our tasks. (rev)	Developer (7 point Likert)	Lewis et al. (2003)
External Software Quality	The capabilities of the software meet the needs of the team's customers (COMPANY internal or external). Overall, the team's software contributes to COMPANY's reputation as a high quality software company. The team delivers software that fully covers the requested functionality. The software the team delivers meets technical requirements.	Product Manager (10 point frequency (0%-90%))	Developed based on 15 interviews
Internal Software Quality	The team complies with done criteria. The software code is maintainable. The software code is clean (e.g., naming, structure, readability, formatting).	Product Manager (10 point frequency (0%-90%))	

References

Austin, J. R. 2003. "Transactive Memory in Organizational Groups: The Effects of Content, Consensus, Specialization, and Accuracy on Group Performance," *Journal of Applied Psychology* (88:5), pp. 866–878.

Bacchelli, A., and Bird, C. 2013. "Expectations, Outcomes, and Challenges of Modern Code Review," in *Proceedings of the International Conference on Software Engineering*, San Francisco, CA, USA, pp. 712–721.

Beller, M., Bacchelli, A., Zaidman, A., and Juergens, E. 2014. "Modern Code Reviews in Open-source Projects: Which Problems Do They Fix?," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, pp. 202–211.

Bliese, P. D. 2000. "Within-Group Agreement, Non-Independence, and Reliability: Implications for Data Aggregation and Analysis," in *Multilevel Theory, Research, and Methods in Organizations: Foundations, Extensions, and New Direction*, K. J. Klein S. W. J. Kozlowski (eds.), San Francisco: Jossey-Bass, pp. 349–381.

Bollen, K. A. 2002. "Latent Variables in Psychology and the Social Sciences," *Annual Review of Psychology* (53:1), pp. 605–634.

Bosu, A., and Carver, J. C. 2014. "Impact of Developer Reputation on Code Review Outcomes in OSS Projects: An Empirical Investigation," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Torino, Italy, pp. 33–42.

Cenfetelli, R. T., and Bassellier, G. 2009. "Interpretation of Formative Measurement in Information Systems Research," *MIS Quarterly* (33:4), pp. 689–707.

Child, J. T., and Shumate, M. 2007. "The Impact of Communal Knowledge Repositories and People-Based Knowledge Management on Perceptions of Team Effectiveness," *Management Communication Quarterly* (21:1), pp. 29–54.

Chin, W. 1998. "The Partial Least Squares Approach to Structural Equation Modeling," *Modern Methods for Business Research* (295:2), pp. 295–336.

Choi, S. Y., Lee, H., and Yoo, Y. 2010. "The Impact of Information Technology and Transactive Memory Systems on Knowledge Sharing, Application, and Team Performance: A Field Study," *MIS Quarterly* (34:4), pp. 855–870.

Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. 2012. "Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository," in *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pp. 1277–1286.

Daniel, S., Agarwal, R., and Stewart, K. 2012. "The Effects of Diversity in Global, Distributed Collectives: A Study of Open Source Project Success," *Information Systems Research* (24:2), pp. 312–333.

Davern, M., Shaft, T., and Te'eni, D. 2012. "Cognition Matters: Enduring Questions in Cognitive IS Research," *Journal of the Association for Information Systems* (13:4), pp. 273–314.

Edmondson, A. C., Dillon, J. R., and Roloff, K. S. 2007. "Three Perspectives on Team Learning," *The Academy of Management Annals* (6), pp. 269–314.

Ellis, A., Porter, C., and Wolverton, S. 2008. "Learning to Work Together: An Examination of Transactive Memory System Development in Teams," in *Work Group Learning: Understanding, Improving, and Assessing How Groups Learn in Organizations*, M. London V. I. Sessa (eds.), New York, NY: Erlbaum Associates, pp. 91–115.

Espinosa, J. A., Slaughter, S. A., Kraut, R. E., and Herbsleb, J. D. 2007. "Team Knowledge and Coordination in Geographically Distributed Software Development," *Journal of Management Information Systems* (24:1), pp. 135–169.

Fægri, T. E., Dybå, T., and Dingsøyr, T. 2010. "Introducing Knowledge Redundancy Practice in Software Development: Experiences with Job Rotation in Support Work," *Information and Software Technology* (52:10), pp. 1118–1132.

Fagan, M. 1986. "Advances in Software Inspections," *IEEE Transaction on Software Engineering* (:7), pp. 744–751.

Faraj, S., and Sproull, L. 2000. "Coordinating Expertise in Software Development Teams," *Management Science* (46:12), pp. 1554–1568.

Gefen, D., Rigdon, E. E., and Straub, D. 2011. "An Update and Extension to SEM Guidelines for Administrative and Social Science Research," *MIS Quarterly* (35:2), pp. III–XIV.

Goodhue, D. L., Thompson, R., and Lewis, W. 2013. "Why You Shouldn't Use PLS: Four Reasons to Be Uneasy about Using PLS in Analyzing Path Models," in *Proceedings of the 46th Hawaii International Conference on System Sciences*, pp. 4739–4748.

Gray, P. H. 2000. "The Effects of Knowledge Management Systems on Emergent Teams: Towards a Research Model," *The Journal of Strategic Information Systems* (9:2–3), pp. 175 – 191.

Griffith, T. L., and Sawyer, J. E. 2010. "Multilevel Knowledge and Team Performance," *Journal of Organizational Behavior* (31:7), pp. 1003–1031.

Guinan, P. J., Cooprider, J. G., and Faraj, S. 1998. "Enabling Software Development Team Performance During Requirements Definition: A Behavioral Versus Technical Approach," *Information Systems Research* (9:2), pp. 101–125.

Gupta, N., and Hollingshead, A. B. 2010. "Differentiated versus Integrated Transactive Memory Effectiveness: It Depends on the Task," *Group Dynamics: Theory, Research, and Practice* (14:4), pp. 384 –398.

Hair, J., Black, W., Babin, B., and Anderson, R. 2009. *Multivariate Data Analysis*, Upper Saddle River, NJ: Prentice Hall.

Hollingshead, A. B. 1998. "Communication, Learning, and Retrieval in Transactive Memory Systems," *Journal of Experimental Social Psychology* (34:5), pp. 423–442.

Hollingshead, A. B. 2000. "Perceptions of Expertise and Transactive Memory in Work Relationships," *Group Processes & Intergroup Relations* (3:3), pp. 257–267.

Hollingshead, A. B., Gupta, N., Yoon, K., and Brandon, D. P. 2011. "Transactive Memory Theory and Teams: Past, Present, and Future," in *Theories of Team Cognition: Cross-Disciplinary Perspectives*, E. Salas, S. M. Fiore and M. Letsky (eds.): Taylor & Francis, pp. 421–455.

Ilgen, D. R., Hollenbeck, J. R., Johnson, M., and Jundt, D. 2005. "Teams in Organizations: From Input-Process-Output Models to IMOI Models," *Annual Review of Psychology* (69), pp. 517–543.

Janz, B., and Prasarnphanich, P. 2009. "Freedom to Cooperate: Gaining Clarity Into Knowledge Integration in Information Systems Development Teams," *IEEE Transactions on Engineering Management* (56:4), pp. 621–635.

Kock, N. 2013. *WarpPLS 4.0 User Manual*, Laredo, TX, USA: ScriptWarp Systems.

Kock, N., and Lynn, G. S. 2012. "Lateral Collinearity and Misleading Results in Variance-Based SEM: An Illustration and Recommendations," *Journal of the Association for Information Systems* (13:7), pp. 546–580.

Kotlarsky, J., and Oshri, I. 2005. "Social Ties, Knowledge Sharing and Successful Collaboration in Globally Distributed System Development Projects," *European Journal of Information Systems* (14:1), pp. 37–48.

Kotlarsky, J., Scarbroug, H., and Oshri, I. 2014. "Coordinating Expertise across Knowledge Boundaries in Offshore-Outsourcing Projects: The Role of Codification," *MIS Quarterly* (38:2), pp. 607–627.

Kozlowski, S., and Klein, K. 2000. "A Multilevel Approach to Theory and Research in Organizations: Contextual, Temporal, and Emergent Processes," in *Multilevel Theory, Research, and Methods in Organizations: Foundations, Extensions, and New Directions*, K. J. Klein S. W. J. Kozlowski (eds.), San Francisco, CA: Jossey-Bass, pp. 3–90.

Larman, C., and Vodde, B. 2010. *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*, Boston, MA: Pearson Education.

Lee, G. K., and Cole, R. E. 2003. "From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development," *Organization Science* (14:6), pp. 633–649.

Lewis, K. 2003. "Measuring Transactive Memory Systems in the Field: Scale Development and Validation," *Journal of Applied Psychology* (88:4), pp. 587–604.

Lewis, K. 2004. "Knowledge and Performance in Knowledge-Worker Teams," *Management science* (50:11), pp. 1519–1533.

Lewis, K., Belliveau, M., Herndon, B., and Keller, J. 2007. "Group Cognition, Membership Change, and Performance: Investigating the Benefits and Detriments of Collective Knowledge," *Organizational Behavior and Human Decision Processes* (103:2), pp. 159–178.

Lewis, K., and Herndon, B. 2011. "Transactive Memory Systems: Current Issues and Future Research Directions," *Organization Science* (22), pp. 1254–1265.

Liang, J., and Mizuno, O. 2011. "Analyzing Involvements of Reviewers Through Mining A Code Review Repository," in *Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, Nara, Japan.

Liang, T.-P., Jiang, J., Klein, G., and Liu, J.-C. 2010. "Software Quality as Influenced by Informational Diversity, Task Conflict, and Learning in Project Teams," *IEEE Transactions on Engineering Management* (57:3), pp. 477 –487.

Lin, T.-C., Hsu, J. S.-C., Cheng, K.-T., and Wu, S. 2011. "Understanding the Role of Behavioural Integration in ISD teams: An Extension of Transactive Memory Systems Concept," *Information Systems Journal* (22:3), pp. 211–234.

Majchrzak, A., Wagner, C., and Yates, D. 2013. "The Impact of Shaping on Knowledge Reuse for Organizational Improvement with Wikis," *MIS Quarterly* (37:2), pp. 455–469.

Mangalaraj, G., Nerur, S., Mahapatra, R., and Price, K. H. 2014. "Distributed Cognition in Software Design: An Experimental Investigation of the Role of Design Patterns and Collaboration," *MIS Quarterly* (38:1), pp. 249–274.

Maruping, L. M., Zhang, X., and Venkatesh, V. 2009. "Role of Collective Ownership and Coding Standards in Coordinating Expertise in Software Project Teams," *European Journal of Information Systems* (18:4), pp. 355–371.

Mc Connell, S. 2004. *Code Complete - A Practical Handbook of Software Construction*: Microsoft Press, 2. edition.

McIntosh, S., Kamei, Y., Adams, B., and Hassan, A. E. 2014. "The Impact of Code Review Coverage and Code Review Participation on Software Quality: A Case Study of the Qt, VTK, and ITK Projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, New York, NY, USA, pp. 192–201.

Müller, M. M. 2004. "Are Reviews an Alternative to Pair Programming?," *Empirical Software Engineering* (9:4), pp. 335–351.

Müller, M. M. 2005. "Two Controlled Experiments Concerning the Comparison of Pair Programming to Peer Review," *Journal of Systems and Software* (78), pp. 166–179.

Nevo, D., Benbasat, I., and Wand, Y. 2012. "Understanding Technology Support for Organizational Transactive Memory: Requirements, Application, and Customization," *Journal of Management Information Systems* (28:4), pp. 69–98.

Nevo, D., and Ophir, R. 2012. "Transactive Memory and Its Application in IS Research," in *Information Systems Theory*, Y. K. Dwivedi, M. R. Wade and S. L. Schneberger (eds.): Springer, chapter 3, pp. 41–58.

Nevo, D., and Wand, Y. 2005. "Organizational Memory Information Systems: A Transactive Memory Approach," *Decision Support Systems* (39:4), pp. 549–562.

Nidumolu, S. 1995. "The Effect of Coordination and Uncertainty on Software Project Performance: Residual Performance Risk as an Intervening Variable," *Information Systems Research* (6:3), pp. 191–219.

Oshri, I., van Fenema, P., and Kotlarsky, J. 2008. "Knowledge Transfer in Globally Distributed Teams: The Role of Transactive Memory," *Information Systems Journal* (18:6), pp. 593–616.

Pangsakulyanont, T., Thongtanunam, P., Port, D., and Iida, H. 2014. "Assessing MCR Discussion Usefulness Using Semantic Similarity," in *6th International Workshop on Empirical Software Engineering in Practice*, pp. 49–54.

Peltokorpi, V. 2004. "Transactive Memory Directories in Small Work Units," *Personnel Review* (33:4), pp. 446–467.

Peltokorpi, V. 2008. "Transactive Memory Systems," *Review of General Psychology* (12:4), pp. 378.

Petter, S., Straub, D., and Rai, A. 2007. "Specifying Formative Constructs in Information Systems Research," *MIS Quarterly* (31:4), pp. 623–656.

Phongpaibul, M., and Boehm, B. 2006. "An Empirical Comparison Between Pair Development and Software Inspection in Thailand," in *Proceedings of the International Symposium on Empirical Software Engineering 2006*, Rio de Janeiro, Brazil.

Podsakoff, P., MacKenzie, S., Lee, J., and Podsakoff, N. 2003. "Common Method Biases in Behavioral Research: A Critical Review of the Literature and Recommended Remedies," *Journal of Applied Psychology* (88:5), pp. 879–903.

Ren, Y., and Argote, L. 2011. "Transactive Memory Systems 1985–2010: An Integrative Framework of Key Dimensions, Antecedents, and Consequences," *The Academy of Management Annals* (5:1), pp. 189–229.

Rigby, P. 2011. "Understanding Open Source Software Peer Review: Review Processes, Parameters and Statistical Models, and Underlying Behaviours and Mechanisms," PhD thesis, University of Ottawa, Department of Computer Science.

Rigby, P., Cleary, B., Painchaud, F., Storey, M.-A., and German, D. 2012. "Contemporary Peer Review in Action: Lessons from Open Source Development," *IEEE Software* (29:6), pp. 56–61.

Rigby, P., German, D., and Storey, M. 2008. "Open source software peer review practices: a case study of the apache server," in *Proceedings of the International Conference on Software Engineering*, Leipzig, Germany, pp. 541–550.

Rigby, P. C., and Bird, C. 2013. "Convergent Contemporary Software Peer Review Practices," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pp. 202–212.

Rigby, P. C., and Storey, M.-A. 2011. "Understanding Broadcast Based Peer Review on Open Source Software Projects," in *Proceedings of the International Conference on Software Engineering*, Honolulu, HI, USA, pp. 541–550.

Ringle, C. M., Sarstedt, M., and Straub, D. 2012. "A Critical Look at the Use of PLS-SEM in MIS Quarterly," *MIS Quarterly* (36:1), pp. iii–xiv.

Rönkkö, M., and Evermann, J. 2013. "A Critical Examination of Common Beliefs About Partial Least Squares Path Modeling," *Organizational Research Methods* (16:3), pp. 425–448.

Rulke, D. L., and Rau, D. 2000. "Investigating the Encoding Process of Transactive Memory Development in Group Training," *Group & Organization Management* (25:4), pp. 373–396.

Ryan, S., and O'Connor, R. V. 2013. "Acquiring and Sharing Tacit Knowledge in Software Development Teams: An Empirical Study," *Information and Software Technology* (55:9), pp. 1614–1624.

Salleh, N., Mendes, E., and Grundy, J. C. 2011. "Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review," *IEEE Transactions on Software Engineering* (37:4), pp. 509–525.

Sauer, C., Jeffery, D. R., Land, L., and Yetton, P. 2000. "The effectiveness of software development technical reviews: A behaviorally motivated program of research," *IEEE Transactions on Software Engineering* (26:1), pp. 1–14.

Schwaber, K., and Beedle, M. 2002. *Agile Software Development with Scrum*, volume 18, Upper Saddle River, NJ: Prentice Hall.

Sommerville, I. 2012. *Software Engineering*, Munich: Pearson Education, 9. edition.

Su, C. 2012. "Who Knows Who Knows What in the Group? The Effects of Communication Network Centralities, Use of Digital Knowledge Repositories, and Work Remoteness on Organizational Members' Accuracy in Expertise Recognition," *Communication Research* (39:5), pp. 614–640.

Sutherland, A., and Venolia, G. 2009. "Can Peer Code Reviews Be Exploited for Later Information Needs?," in *31st International Conference on Software Engineering-Companion Volume, 2009.*, pp. 259–262.

Thongtanunam, P., Tantithamthavorn, C., Kula, R. G., Yoshida, N., Iida, H., and Matsumoto, K. 2015. "Who Should Review My Code? A File Location-Based Code-Reviewer Recommendation Approach for Modern Code Review," in *Proceedings of The 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering*, Montreal, Canada.

Thongtanunam, P., Yang, X., Yoshida, N., Kula, R., Camargo Cruz, A., Fujiwara, K., and Iida, H. 2014. "ReDA: A Web-Based Visualization Tool for Analyzing Modern Code Review Dataset," in *IEEE International Conference on Software Maintenance and Evolution*, Sept, pp. 605–608.

Tiwana, A., and Mclean, E. 2005. "Expertise Integration and Creativity in Information Systems Development," *Journal of Management Information Systems* (22:1), pp. 13–43.

Tomayko, J. E. 2002. "A Comparison of Pair Programming to Inspections for Software Defect Reduction," *Computer Science Education* (12:3), pp. 213–222.

Wang, J., and Carroll, J. 2011. "Behind Linus's Law: A Preliminary Analysis of Open Source Software Peer Review Practices in Mozilla and Python," in *International Conference on Collaboration Technologies and Systems*, pp. 117–124.

Wegner, D. M. 1987. "Transactive Memory: A Contemporary Analysis of the Group Mind," in *Theories of Group Behavior*, B. Mullen G. R. Goethals (eds.), New York, NY: Springer, pp. 185–208.

Wegner, D. M., Giuliano, T., and Hertel, P. T. 1985. "Cognitive Interdependence in Close Relationships," in *Compatible and Incompatible Relationships*, Berlin: Springer, pp. 253–276.

Wetzels, M., Odekerken-Schroder, G., and van Oppen, C. 2009. "Using PLS Path Modeling for Assessing Hierarchical Construct Models: Guidelines and Empirical Illustration," *MIS Quarterly* (33:1), pp. 177–195.

Winkler, D., and Biffl, S. 2006. "An Empirical Study on Design Quality Improvement from Best-Practice Inspection and Pair Programming," in *Proceedings of the 7th International Conference on Product-Focused Software Process Improvement*, Amsterdam, The Netherlands, pp. 319–333.

Yang, X. 2014. "Social Network Analysis in Open Source Software Peer Review," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Hong Kong, China, pp. 820–822.

Yuan, Y. C., Fulk, J., and Monge, P. R. 2007. "Access to Information in Connective and Communal Transactive Memory Systems," *Communication Research* (34:2), pp. 131–155.