UNIVERSITY OF LEEDS

This is a repository copy of *Risks to Effective Knowledge Sharing in Agile Software Teams: A Model for Assessing and Mitigating Risks*.

White Rose Research Online URL for this paper:
http://eprints.whiterose.ac.uk/144697/

Version: Accepted Version

Article:
Ghobadi, S and Mathiassen, L (2017) Risks to Effective Knowledge Sharing in Agile Software Teams: A Model for Assessing and Mitigating Risks. Information Systems Journal, 27 (6). pp. 699-731. ISSN 1350-1917

https://doi.org/10.1111/isj.12117

© 2016 John Wiley & Sons Ltd. This is the peer reviewed version of the following article: Ghobadi, S., and Mathiassen, L. (2017) Risks to Effective Knowledge Sharing in Agile Software Teams: A Model for Assessing and Mitigating Risks. Info Systems J, 27: 699–731, which has been published in final form at https://doi.org/10.1111/isj.12117. This article may be used for non-commercial purposes in accordance with Wiley Terms and Conditions for Self-Archiving. Uploaded in accordance with the publisher's self-archiving policy.

Reuse
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

White Rose university consortium
Universities of Leeds, Sheffield & York

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

A Model for Assessing and Mitigating Knowledge Sharing Risks in Agile Software Development

Shahla Ghobadi, The University of Manchester
Lars Mathiassen, Georgia State University

What happens is not as important as how you react to what happens. Ellen Glasgow

Abstract

We present an empirically-grounded and theoretically-informed model for the assessment and mitigation of risks to effective knowledge sharing in agile development. The model is anchored in empirical insights from four agile projects across two software companies and in extant research on risk-strategy analysis and knowledge sharing in software development. We develop the model as part of the long-standing tradition of presenting risk management models dedicated to specific issues in software development and confirm its practical usefulness in one of the software companies studied. The model offers concepts and processes to assess a project's knowledge sharing risk profile and articulate an overall resolution strategy plan to mitigate the risks. The results highlight how different knowledge sharing risk management profiles can lead to different project performance outcomes. We conclude with a discussion of research opportunities that the results offer software development scholarship.

Keywords: Agile, software development, knowledge sharing, knowledge management, risk management, qualitative research, grounded theory

Introduction

The use of agile practices such as eXtreme programming and scrum improves team agility and helps address process inefficiencies common in plan-driven software development (Holmström et al., 2006, McAvoy et al., 2012, Highsmith, 2009). A fundamental concept in agile development is effective sharing of high-quality information, know-how, ideas, suggestions, skills, and expertise among individuals (Ghobadi & D'Ambra, 2013). For example, scrum requires user representatives, product owners, developers and managers to engage in iterative cycles, address development challenges, and explore product opportunities (Nerur & Balijepally, 2007, Carmel et al., 2010, Chakraborty & Sarker, 2010). Several barriers may, however, pose risks to effective knowledge sharing in agile development (Ghobadi & Mathiassen, 2015). Some examples include diverse working and discipline-related backgrounds among team members (Corvera Charaf et al., 2012), different time zones and physical distance between team members (Conboy et al., 2010, Dorairaj et al., 2012, Gupta & Bajwa, 2012), and insufficient planning and documentation (Karlsen et al., 2011, Conboy & Morgan, 2011). Inevitable knowledge sharing barriers prompt calls to pay closer attention to understanding the risks they pose to software practices and in turn to develop strategies that help mitigate those risks. Although the extant literature recognizes these barriers (Ghobadi & Mathiassen, 2015), there is limited knowledge and no comprehensive approach on how agile development teams can manage these risks. This lack of research can be attributed to existing views that link 'formalized management approaches' to going against the agile philosophy of 'people over processes' and to stifling the positive benefits of risk taking behaviors (Dalcher, 2002). More recently, however, research has proclaimed the importance of seeking a balanced view in which the strengths of both agile and plan-driven approaches are leveraged (Boehm & Turner, 2003). A well-respected plan-driven approach is to adopt risk management to assess and mitigate risks related to software development (Boehm & Turner, 2003, Boehm, 1991). Risk management is also helpful for creating shared mental models across stakeholders and for supporting collective decision making (Lyytinen et al., 1998). Hence, based on a risk management approach this study concentrates on the following research question: how can agile development teams systematically assess and mitigate risks to effective knowledge sharing?

We use the term *'agile development teams'* to refer to contemporary software teams that actively use agile practices in their development efforts. We then define 'risks to effective knowledge sharing' as barriers that (with some likelihood) may adversely affect (with some loss) effective knowledge sharing in agile development. In addressing the research question, we rely on a grounded theory approach (Glaser & Strauss, 1977). Specifically, we complement empirical data collected from four agile projects with (i) key findings within the agile literature (Conboy et al., 2010, Vidgen & Wang, 2009, McAvoy et al., 2012, Ramesh et al., 2010, Conboy & Morgan, 2011, Ghobadi & Mathiassen, 2015), and (ii) insights from risk management research (Davis, 1982, Persson et al., 2009, Iversen et al., 2004). The result is an empirically-grounded and theoretically-informed model for assessing and mitigating risks to effective knowledge sharing in agile development. This study presents three theoretical contributions.

First, our risk management model synthesizes a generic list of 37 risk items and 31 resolution actions. This list covers an extensive set of knowledge sharing risks and resolutions that are not integrated in existing software risk management frameworks (Persson et al., 2009, Davis, 1982). The model contributes categories, concepts, and processes that are helpful to both qualitative and quantitative research studying communication-related issues in software contexts. Second, our model offers a systematic approach to risk management in agile software development. Specifically, it contributes to the long-standing tradition of developing risk management models in software development (Boehm, 1991, Barki et al., 1993, Baskerville & Stage, 1996) with heuristics to assess risks to effective knowledge sharing, to identify and prioritize resolution actions to mitigate them, and to articulate an overall resolution strategy plan. Third, our results suggest the high performing projects, more than the low performing ones, tended to address risks more effectively by taking bolder initiatives and applying more resolution actions relative to existing risks. This finding concurs with prior software development research (Barki et al., 2001), suggesting that different project risk management profiles can lead to different project performance outcomes.

The remainder of the article is organized as follows. We begin by detailing theoretical background and research methodology, including data collection and analysis procedures. We explicate research findings for each development project followed by complementing the cross-case analysis with the extant literature. Next, we present the proposed model and the results of its practical evaluation. We conclude by discussing implications for theory and practice and outlining avenues for future research.

Theoretical Background

Researchers have long studied the intensive, collaborative, and knowledge-intensive processes through which software emerges (Ghobadi, 2015). We, therefore, know several barriers, such as diverse social identities, cross-functionality of team members, coordination challenges across distributed sites and motivational factors that may complicate knowledge sharing in software teams. New software trends revolving around agile development have generated renewed interest in this area as well (Dybå & Dingsøyr, 2008). Specifically, agile practices are based on principles that focus on welcoming change, working software, and continuous introspection (Williams, 2012). Agile practices are set to improve communication and knowledge sharing in software contexts. For example, postmortem reviews encourage team members to share and learn from good and bad project experiences (Dingsøyr & Hanssen, 2003). Another example is pair programming that helps foster sharing of embedded knowledge (Bellini et al., 2005, Ghobadi et al., 2015). Implementing agile practices may, however, pose unintended risks to knowledge sharing, putting agile teams at the risk of losing requisite capabilities. For example, frequent releases are recommended to facilitate knowledge sharing across stakeholders (Lippert et al., 2003). However, over-communication between the team and customers exposes software teams to the risk of losing agility (Vidgen & Wang, 2009). Also, including customer representatives at sprint planning sessions helps streamline communication with the client and facilitate organic change (Karlsen et al., 2011). However, this practice can reduce the available time for sharing ideas outside the team (Conboy & Morgan, 2011). In addition, customer representatives may generate major reworks for software teams and make it difficult to commit enough time to knowledge sharing at later stages of development (Batra, 2009).

In summary, agile development teams should pay special attention to identify barriers to effective knowledge sharing and to mitigate the risks they pose to development contexts. There is, however, limited knowledge and no comprehensive approach on how agile development teams can manage knowledge sharing risks

(Ghobadi & Mathiassen, 2015). Traditionally, risk management approaches are used to identify and assess software development risks (Lyytinen et al., 1998, Mcfarlan, 1981). These approaches encapsulate the key elements of risks (risky incidents), resolution actions (possibly relevant actions), and heuristics (guidelines for assessing risks and linking them to appropriate resolution actions). As an example, risk-strategy analysis models offer a stepwise process that links 'detailed analysis of risks' to 'an overall risk management strategy' (Persson et al., 2009, Iversen et al., 2004, Davis, 1982); The underlying idea of these models is consistent with our interest in developing a detailed approach to identifying knowledge sharing barriers and mitigating the risks they may pose to development practices.

Extant literature has developed risk management models targeting several aspects of software development such as implementation risks (Lyytinen, 1987), project portfolio risks (Earl, 1987), requirement management risks (Ramesh et al., 2010, Davis, 1982), distributed development risks (Persson et al., 2009), and prototype development risks (Baskerville & Stage, 1996). Despite their diversity, the majority of these studies only offer ad-hoc assessment of risks and possible resolution actions. There are scarce examples that offer systematically-developed list of risks and resolution actions (Iversen et al., 2004). In addition, there are no models for managing 'knowledge sharing risks' in software projects. Addressing these gaps, our study develops an intellectual tool and theoretical implications that help understand and manage the complex knowledge sharing risks in agile development contexts.

Research Method

The grounded theory approach is well-suited to building theoretical insights in an area where limited understanding exists and where we can respond flexibly to new empirical discoveries (Eisenhardt, 1989). We use this approach in the following manner. First, we conduct a multisite case study to invoke new insights for the assessment and mitigation of risks to effective knowledge sharing in agile development (Glaser & Strauss, 1977, Eisenhardt, 1989). Second, we complement the grounded understanding with insights from the extant literature. Third, we refine the resulting model by examining its practical usefulness in one software company. The following sections describe data collection and analysis processes.

Data Collection

This study is part of a larger research project on knowledge sharing in agile development (Ghobadi & Mathiassen, 2015). Therefore, we rely on an overlapping yet expanded set of data compared to our earlier study on knowledge sharing in agile teams. We collected empirical data through several sessions of iterative and semi-structured interviews over twelve months across two medium-size software companies. The companies, referred to as Alpha and Beta, are based in Australia (~100 employees each). They are leading companies in building software for financial companies, climate change centers, and biomedical institutions.

The key theme of their work is developing software that allows capturing, storing, sharing, and analyzing data. Alpha and Beta characterize their development approach by advocating 'user stories', 'product backlog', 'working software', and 'responsiveness to change'. In their website and contracts' documents, they highlight agile practices such as 'sprint planning', 'stand-up meetings', and 'pair programming'. In fact, the companies emphasize using agile practices to remain relevant in the software industry and to increase agility in their work. Besides, they take pride in not being 'textbook' agile, but rather having expertise in adapting agile practices to remain flexible and agile in practice. Differences between documented agile practices and their practical use are quite common (Vidgen & Wang, 2009, Fitzgerald et al., 2000, Fitzgerald et al., 2002). For example, if the client cannot provide on-site customer representatives as required in pure agile contexts, the companies find a way to establish an effective distributed team and work closely with customers.

We consulted the development manager of each company to provide entry for conducting the fieldwork. The development managers helped recruit eight interviewees from two recent agile projects that were completed within the last three months. We sought advice from development managers for two reasons. First, they hold a comprehensive overview of projects, their performance, and the individuals working on them. Second, this approach enables immediate legitimacy and credibility to the research, which helped significantly during the interview sessions. We interviewed individuals holding the four roles of 'project manager', 'developer', 'tester', and 'user representative'. We based the selection of these roles on prior research on key roles in software development (Newman & Robey, 1992, Barki & Hartwick, 2001) as well as key roles in medium to large agile teams (Sutherland, 2005). In addition, the four roles are fundamental in Alpha and Beta's agile practices. Specifically, (i) project owners take ownership of the product (they are labeled as project managers in this study), (ii) developers produce code, (iii) quality assurance engineers (QA) test the product, (iv) client representatives serve as end user representatives, and at times user interface or user experience designers conduct user experience design. Further information on the key roles is provided in our earlier study (Ghobadi & Mathiassen, 2015).

We asked the development manager of each company to select one high performing and one low performing project. The polar sampling approach (Eisenhardt & Graebner, 2007) helped generate additional insights related to different patterns of risks, resolution actions, and project outcomes. During data collection, developers, project managers, tester and user representatives expressed similar views on the performance of the selected projects. Table 1 presents the demographics of the four projects. An official email from each development manager introduced the research project to the relevant staff. The official emails highlighted the academic focus of the research, and the interviewees were ensured confidentiality. The sixteen in-depth interview sessions, lasting between 45 minutes and an hour, were taped and transcribed. We first asked about barriers that pose risks to effective knowledge sharing practices. Appendix 1 provides the interview questions for identifying risks and further explanations are provided in our earlier work (Ghobadi & Mathiassen, 2015). We then adopted a flexible and open style of conversation to identify resolution actions. The researcher asked the initial questions of 'please elaborate how this barrier inhibited *effective knowledge sharing*' and *'which steps were taken to deal with it?'* The following questions varied depending on the interviewees' responses and their ability to elaborate on the subject. For example, the researcher asked: '*how effective was this action'* and *'why did the team not take an action?'* As a final check, questions pointing into specific risks and resolution actions afforded interviewees a closing opportunity to report any item that might have been missed or required further explanation. When interviewees revealed opposing views, further investigation such as informal follow-up interviews helped explore differing perspectives and arrive at a richer understanding. Following an initial data analysis, follow-up interviews resolved ambiguities and validated interpretive accuracy and credibility (Guba & Lincoln, 1985). For instance, there were cases in which a risk and a related resolution action were mentioned, yet we needed to confirm the use of that specific resolution action for addressing the related risk. A total of 36 interviews helped develop a detailed understanding of the 'risks' and 'implemented resolution actions' in the four studied agile projects.

Table 1. Project Characteristics

Item	Alpha One	Alpha Two	Beta One	Beta Two
Project Duration	4 months	6 months	9.5 months	2 months
Team Size	7 members	10 members	10 members	12 members
Team Members Age	33.8 years	35.2 years	36.3 years	40.1 years
Team Members Education	Undergraduate: 50% Postgraduate: 50%	Undergraduate: 75% Postgraduate: 25%	Undergraduate: 75% Postgraduate: 25%	Undergraduate: 75% Postgraduate: 25%
Team Members Software Experience	5.8 years	8.6 years	10.5 years	9.0 years
Team Members Company Experience	4.5 years	3.5 years	4.5 years	5.0 years
Agile practices	Design meetings, stand ups, retrospectives, pair programming, burn down charts.			
Sprints Length	Varied between 2-3 weeks			

Data Analysis

Data analysis progressed in four steps. The first two steps involved within-case and cross-case analyses of the risks and resolution actions in each of the four projects. The next two steps included development of a risk management model and evaluating its practical usefulness. Each step is explained below.

1. **Within-case Analyses**: One researcher read all the interview transcripts, grouped frequently mentioned words together, and generated a list of codes that correspond to (i) risk items to effective knowledge sharing and (ii) resolution actions to mitigate those risks. Another researcher verified face validity, parsimony, and coverage of the coding scheme. We also conducted collaborative code training between

us to minimize coding biases by the researcher. Next, we coded all transcripts to identify and link risk items and resolution actions. For identifying 'risk items' and 'resolution actions', we systematically looked for 'barriers that may have adversely affected knowledge sharing practices' and 'actions that were implemented to mitigate the risks that the barriers posed'. For example, a user representative argued that developers tried to be flexible with the client's situation to resolve their initial lack of communication regarding time requirements. Thus, *'being flexible with client's situation*' was noted as a resolution action for mitigating the risk that '*lack of communication of time requirements to the client'* posed to effective knowledge sharing. In another example, a user representative explained that training workshops helped them understand how agile teams work, the role of prototypes, and how important it is to communicate end user requirements particularly in the absence of a good prototype. Thus, *'runni*ng workshops to improve client *understanding of agile processes'* was noted as a resolution action for mitigating the risk that '*lack of a good prototype for communicating requirements'* posed to effective knowledge sharing. Sample codes are provided in Appendix 2. To categorize risk items into risk areas, we utilized the existing conceptualization of knowledge sharing barriers, including: (i) team diversity, (ii) team perceptions, (iii) team capabilities, (iv) project communication, (v) project organization, (vi) project technology, and (vii) project setting (Ghobadi & Mathiassen, 2015). The final coding progressed as follows. One researcher coded the first eight transcripts. The other researcher checked the validity of the codes. We calculated Scott's pi at an acceptable level of 0.86 (Scott, 1955). The researcher finalized the process by coding the next eight transcripts. Summaries of within-case analysis are discussed in the Results section and summarized in Appendix 3.

2. **Cross-case Analysis:** We removed redundant items in within-case analysis tables, merged similar ones, and took initial steps away from company specific jargon to generate more general findings (Lee & Baskerville, 2003). For example, we consistently adopted an imperative form for presenting resolutions to emphasize their action orientation (e.g., delegate, emphasize, create, or relocate). As an example, we used the term '*recruit developers with a combination of IT and business knowledge'* to refer to the resolution action *'make experienced developers available for translating busi*ness needs into technical *terms'*. This resulted in identifying a total of 31 risk items and 20 resolution actions. We then complemented the empirical results with extant literature on knowledge sharing in agile development. As an example, we included the recommendation to '*routinize exploration in development teams'* to highlight the importance of allocating resources to encourage team members to search for and share new ideas (Vidgen & Wang, 2009). In total, six risk items and eleven resolution actions were added. This expanded the findings to 37 risk items and 31 resolution actions. The results are detailed in the Cross-case Analysis section.

3. **Model Development:** We began by categorizing the identified 31 resolution actions. We carefully went through each resolution action and the related quotes to cluster together actions that correspond to similar issues. For example, all the following four actions concentrate on improving working relationships within team and with client: (i) Leverage team diversity through cross-team observation and close team member collaboration, (ii) Promote positive relationships across stakeholders, (iii) Build collaborative relationship with IT team in client organization, and (iv) Leverage positive relationships between client representatives and client management. We, therefore, grouped them under the resolution strategy of Leverage Relationships. Finally, we relied on the vocabulary of risk-strategy analysis approaches to organize the key findings into a risk management model. The key vocabulary items include risk items, risk areas, resolution actions, resolution strategies, heuristics, and stepwise process. The result is detailed in the Model Development section.

4. ***Model Evaluation*:** We examined and refined the risk management model from the last step based on its practical application in Alpha as well as useful comments from academic colleagues. The researcher presented the model to a number of practitioners and asked them to apply it to an ongoing agile development project. The researcher was prepared to answer any questions they had. Observing practitioners and interacting with them generated useful ideas to improve the presentation, wordings, and structure of the model. For example, we found it is easier for participants to put selection boxes beside the risk items and not in another column. The results are detailed in the Model Evaluation section.

Results

Alpha One

Project One in Company Alpha (Alpha One) spent 4 months to develop a system that manages data and metadata associated with textual artifacts from ancient civilizations. The project team included seven members, including project manager, three developers (one had the role of scrum master), tester, user interface designer, and user representative, with an average age of 32 years. According to the development manager, Alpha One proved to be a high performing project because: "Stakeholders got the results they wanted and they are happy with how they worked with the team. Also, I think the stakeholders were realistic and worked with us collaboratively rather than in an adversarial relationship".

Knowledge was shared through face to face channels such as design meetings and daily stand ups as well as technology mediated channels such as Skype and Enterprise-hosted collaborative tools. The interviewees referred to a total of eight risk items in Alpha One. These risk items are categorized into four risk areas. For example, lack of communication of time requirements to the client was considered a risk to effective knowledge sharing. According to the user representative, he was not prepared for intensive knowledge sharing that is crucial in agile development. This was because time requirements were not communicated to him right at the beginning (insufficient communication of time requirements; related to the category of project communication risks): "Most of the people in [my business] have heavy workloads and administrative loads. *I didn't understand how much time would be required of me at the start. Ultimately it* worked well, but it is important that the people that work with agile teams understand how much time they must commit to the project and get prepared for future correspondence" (user representative). The development team responded to this risk by trying to be flexible and understanding. Said the user representative: "It seemed to me there was a clear awareness of the kinds of pressures that people in [my business] have. So when I had difficulty making myself available, they [development team] were very quick to adapt" (user representative).

Another risk item was unfamiliarity of the development team with the built-in coding technology in the legacy system (related to the category of team capabilities risks). Specifically, the development team was committed to provide demonstrations at the end of each two-week sprint. However, developers had to spend considerable time on learning the new coding language. Thus, it was difficult for them to commit enough time on knowledge sharing regarding several other aspects of the project such as innovative coding solutions and new functionalities. The development team addressed this risk by leveraging diverse capabilities of the experienced team members as well as positive relationships among the team members. One developer explained: "It was Ruby on Rails *application, and I didn't know the language. But [developer A] had a lot of* knowledge in this area. So he shared his knowledge with us. He was very helpful" (developer).

Alpha Two

Project Two in Company Alpha (Alpha Two) spent 6 months to develop a web-based system that assists data collection processes of a collaborative neonatal network. The project team included ten members, including project manager, four developers (one had the role of scrum master), tester, two user interface designers, and two user representatives, with an average age of 34 years. According to the development manager, Alpha Two was considered a low performing project because: "We failed to manage their expectations up front. So, *we couldn't possibly have delivered what they wanted wi*th the budget we had. I think they were working *with us in more of a 'you are service provider, you do everything and tell us when you are done', rather than 'we collaborate together to complete the project.'"*

Not surprisingly, Alpha Two faced a broad range of risks to effective knowledge sharing, and knowledge sharing proved to be challenging. The interviewees referred to nineteen risks. These risks are categorized into seven risk areas. For example, the user representative referred to lack of a good prototype for communicating requirements with users as a risk to effective knowledge sharing (related to the category of project technology risks): "People can comment on tangible things. If you show a blank page and ask what you want for y*our interface they can't say* [share knowledge]. They give you very vague answers. You should show at the beginning which functionalities you can build" (user representative). The development team attempted to reduce this risk indirectly by running workshops to improve client understanding of agile processes and characteristics. Said the user representative: "It was good that our team attended five meetings at [Company Alpha] to understand more of what is happening in the software development and agile world" (user representative). She argued that this understanding encouraged them to communicate user requirements in a more detailed, robust, and active manner, mitigating the risk that lack of a good prototype poses to their project's knowledge sharing practices.

Despite these actions, most of the risks were left unaddressed. For example, a tester referred to risks associated with long split sprints that were not carefully addressed (related to the category of project organization risks): "Developers thought three weeks sprint to be better for their focus, because of rotating shifts with an alternative project. But it was recognized that it was difficult for developers moving between *projects. As a tester, I'd be asking questions about a project [from d*evelopers] that they were not working on currently. [User representatives] would see a lot of activity and then suddenly nothing would happen for three weeks." (tester). In addition, the category of team perceptions risks was largely unaddressed. On the one hand, the project manager argued user representatives' inappropriate assumptions about project scope did now allow discussing critical aspects of the project upfront: "They had certain expectations about what they could get with the very small budget they had. It turned out they [client representatives] were not telling us these. Because we are agile and flexible they assumed we are just going to do that" (project manager). On the other hand, the user representative argued that developers' unrealistic and low estimations masked the complexity of the project. Thus, team members did not discuss various possibilities at early stages of the project. In addition, the user representatives and majority of the end users came from a non-English speaking background. A developer pointed to different spoken languages as a risk to effective knowledge sharing. The interviewees, however, did not refer to any action for mitigating this risk. The same developer further explained that Alpha did not discuss this issue during the course of the project, and thus they did not take effective steps to mitigate the risk.

Beta One

Project One in Company Beta (Beta One) spent 9.5 months to develop a financial system that creates maps of stocks and equities based on different types of financial information. The project team included ten members including project manager, business analyst, six developers (one had the role of Scrum master), tester, and user representative, with an average age of 35 years. According to the development manager, Beta One was a high performing project because: "Although the team lost momentum as people moved countries and jobs, we still delivered what we were asked to do. Everyone involved, were satisfied with the process, *and they haven't been burned* out."

The interviewees referred to a total of fifteen risks that are categorized into seven risk areas. For example, time difference between the development team and the client introduced a risk related to the category of team diversity risks: "The client was based in [Country A], so there was only 2 hours a day that we could actually communicate directly. And, for email communication there was always lag, which made certain decisions very slow for an agile project", narrated a developer (developer). The development team responded by applying informal decision making whenever required: *"When* we wanted to get things done, we started to bypass formal decisions about what we put in. Bypassing was the only thing that let sprints move forward. [Formally], if we wanted to change a single word in one of the acceptance criteria, we needed to raise a changing requirement for that and wait for [client] to approve it. We got to the point where we simply made that change and hoped for the best. It is not something to be proud of, but at that point we needed that because it was slowing us down", explained the tester (tester).

Tight sprint schedule with little time for interaction was also noted as a risk to effective knowledge sharing (related to the category of project organization risks). Being a domain-specific project was argued to slow down communication and effective knowledge sharing (related to the category of project setting risks). In response to this risk, the business analyst wrote and communicated clear stories, helping developers understand stories well. According to a developer: "Understanding the data we were dealing with required a lot of domain knowledge that takes time. *But, I don't think that was too much of an actual* communication barrier. I guess when the business analyst wrote a story she worked with the data expert to explain what *needs to be done. As developers we took the stories and we knew what to do, but we didn't necessarily* understand the data. And, that actually worked okay" (developer).

Beta Two

Project Two in Company Beta (Beta Two) spent 2 months to develop a system that automates the integration of data from the stock exchange market. The project team included twelve members, including project manager, nine developers (one had the role of scrum master), tester, and user representative, with an average age of 34 years. Beta Two was considered a low performing project because: "*Project stakeholders didn't* see results fast enough. The team faced major challenges because they lost people constantly and business priorities changed constantly," commented the development manager in Beta.

The interviewees referred to a total of fourteen risks to effective knowledge sharing categorized into seven risk areas as well as different ways in which the team responded to these risks. For instance, 'complex business rules' was raised as a risk to effective knowledge sharing (related to the category of project setting risks). According to the user representative, understanding complex rules was consuming developers' limited time, and so they had much less time to discuss important aspects of the project with themselves and the client: "They [developers] knew what the algorithm was and what the code for it was, but understanding *what the code's ultimate goal* was and spotting errors in the output was hard and time consuming" (user representative). The management team responded to this risk by making the developers who were experienced in the specific domain available for the project." We had [a senior developer] who has the most *experience of these products. We asked him how things work. If he hadn't been available it would have been* much more difficult", argued a developer (developer).

The onsite user representative referred to the risk of development team's lack of motivation (related to the category of team perceptions risks): "This team was sort of plundered of people, so the team that remained *seemed to be less experienced. I think they may have been less motivated, because they didn't feel that the*ir work was as important as what some of the other teams were doing. This certainly did affect the attitudes of the team in sharing knowledge" (user representative). Management agreed and responded to this risk by emphasizing the importance of the project and organizing demos of the working software as a measure of success: "We tried to put forth that these sorts of projects are basically the stuff that have been put off for so long, because no one wanted to do it, but once they got through that work they would be working on new products that all the other people may have wanted to work on. And, we are trying to basically let them be aware that they are in a position that is highly visible in the company and can grow very rapidly" (project manager).

Data analysis showed that the most frequently mentioned action to address risks (resolution action) was to make sure developers who are experienced in the specific domain are assigned to the project (here financial software). This resolution action helped the team members significantly. They were not anymore frustrated in the process of understanding the complex domain in their tight schedule. They could spend time on other aspects of the development such as coding and designing new solutions. Yet, according to the project manager, over time, they realized this resolution action was not an optimal solution: "Because we have [expert] people who have been here for such a long time we are absolutely dependent on them now" (project manager). In addition, some key risks were left unaddressed with adverse consequences. For example, different working backgrounds and personalities of team members were not addressed efficiently. This later proved to be challenging. Said the project manager: "We had a huge amount of discrepancy in terms of *people's knowledge. We had someone who knows the [domain knowledge] over five years. We had people* who had never worked on [this area]. That makes it very difficult for people to swap work and explain concepts daily" (project manager).

Cross-case Analysis

This section integrates within-case empirical findings with insights from extant literature (within-case analyses are provided in Appendix 3). This integration results in a total of 37 risk items and 31 resolution actions. The result of cross-case analysis is summarized in Table 2. First, this table shows the sources for each risk item and resolution action. For example, the risk of 'different speaking languages among members' was raised in Alpha Two and Beta One, or the risk of 'lack of familiarity with agile values and principles' was raised in the extant literature that we integrated into our cross-case analysis (the detailed process is provided in the Data Analysis section) (Conboy et al., 2010). Second, this table relates each resolution action to the risk items it may help address. Third, this table highlights 14 risk items in bold. These 14 risk items refer to those risks that are tightly linked to agile development contexts. As an example, the risk of 'lack of familiarity with agile values and principles' and the risk of 'lack of communication of agile time requirements with client up front' are inherently related to agile development.

Table 2. Identification of Risk Items and Resolution Actions

Knowledge Sharing Risks	**Knowledge Sharing Resolutions**
Team Diversity	
1. Different speaking languages among members *(Alpha Two, Beta One)*	Create and share goals within team *(Beta One)*
2. Different working and disciple-related backgrounds among members *(Beta One, Beta Two)*	Communicate importance of project to team members for key stakeholders and career opportunities *(Beta One)* Create and share goals within team *(Beta One)* Discuss expectations and requirements with client and within team *(Beta Two)* Delegate project responsibilities within team *(Beta One)* Relocate developers to spend more time with each other *(Beta Two)*
3. Different time zones and physical distance between members *(Beta One)*	Support participation and flexibility in project's decision making *(Beta One)*
4. Lack of prior joint working experience in development team *(Beta One, Beta Two)*	Communicate importance of project to team members for key stakeholders and career opportunities *(Beta One)* Create and share goals within team *(Beta One)* Discuss expectations and requirements with client and within team *(Beta Two)* Delegate project responsibilities within team *(Beta One)*
Team Capabilities	
5. Insufficient understanding of business domain and context *(Alpha Two, Beta One, Beta Two)*	Recruit experienced and motivated developers *(Beta Two)* Recruit developers with a combination of IT and business knowledge *(Conboy et al., 2010)* Share historic and current systems documentation across team *(Alpha Two)* Increase developers' business knowledge through client-organized training sessions *(Alpha Two)*
6. Unfamiliarity with development and collaboration technologies *(Alpha One, Beta Two)*	Use pair programming to facilitate learning across development team *(Alpha One)* Relocate developers to spend more time with each other *(Alpha One)*
7. Insufficient and ambiguous requirements *(Beta Two)*	Improve team's agile and social skills through training *(Conboy et al., 2010)*
8. Inadequate social skills *(Alpha Two)*	Improve team's agile and social skills through training *(Conboy et al., 2010)*
9. Lack of familiarity with agile values and principles *(Conboy et al., 2010)*	Support requisite exploration of alternative options *(Vidgen & Wang, 2009)* Engage team in evaluating agile opportunities and challenges *(Ramesh et al.,*

	2010, McAvoy et al., 2012)
10. Lack of IT resources and working experience with software companies in client company *(Alpha One, Alpha Two)*	Provide client with knowledge about agile projects through workshop *(Alpha Two)* Discuss expectations and requirements with client and within team *(Alpha One, Alpha Two)* Analyze client organization dynamics and be open to adapt to changing conditions *(Alpha One, Alpha Two)* Leverage team diversity through cross team observation and close team member collaboration *(Alpha One)* Promote positive relationships across stakeholders *(Alpha One)*
Team Perceptions	
11. Lack of motivation, focus and adaptability in development team *(Alpha Two, Beta One, Beta Two)*	Communicate importance of project to team members for key stakeholders and career opportunities *(Beta One)* Create and share goals within team *(Beta One)* Collect and share successful project stories with team *(Conboy et al., 2010)* Relocate developers to spend more time with each other *(Beta One, Beta Two)* Provide each individual member with 360° feedback *(Conboy et al., 2010)*
12. Inappropriate assumptions about project scope made by client (due to the development team's flexible agile-related approach) *(Alpha Two)*	Discuss expectations and requirements with client and within team *(Alpha Two)*
13. Fear of self-exposure to technical and agile skills deficiencies in development team *(Conboy et al., 2010)*	Help new team members integrate through mentors and incremental responsibilities *(Conboy et al., 2010)* Support requisite exploration of alternative options *(Vidgen & Wang, 2009)* Provide opportunities to raise any concern for discussion in open forums *(Conboy & Morgan, 2011)*
14. Performance evaluation based on technical achievements (related to working software principle) *(Conboy et al., 2010)*	Put high value on mentoring and voluntary contributions in performance evaluations *(Conboy et al., 2010)*
15. Stakeholder neglect of nonfunctional requirement (related to working software principle) *(Ramesh et al., 2010)*	Engage team in evaluating agile opportunities and challenges *(Ramesh et al., 2010, McAvoy et al., 2012)*
Project Communication	
16. Inadequate client availability and participation *(Alpha Two)*	Change the length of split sprints to improve interactions *(Alpha Two)* Analyze client organization dynamics and be open to adapt to changing conditions (*Alpha Two)*
17. Lack of communication of agile time requirements with client up front *(Alpha One, Alpha Two)*	Analyze client organization dynamics and be open to adapt to changing conditions (*Alpha One, Alpha Two)* Support client and team communication with collaboration technologies *(Alpha One)* Share key project information with client representatives using nontechnical language *(Alpha One)* Discuss expectations and requirements with client and within team *(Alpha One)*
18. Lack of concurrence within client team *(Alpha Two)*	Recruit developers with a combination of IT and business knowledge *(Conboy et al., 2010)* Improve team's agile and social skills through training *(Conboy et al., 2010)*
19. Product owner lack of sharing client feedback with development team *(Beta One)*	Discuss expectations and requirements with client and within team *(Beta One)*
Project Organization	
20.Tight sprints schedule with little time for interaction *(Alpha Two, Beta One, Beta Two)*	Create and share goals within team *(Beta One)* Communicate importance of project to team members for key stakeholders and career opportunities *(Beta One)* Recruit experienced and motivated developers *(Beta Two)*
21. Inadequate planning and organization in agile	Document experiences to support planning of future projects *(Beta Two)*

practices *(Alpha Two, Beta One, Beta Two)*	
22. Multitasking and lack of continuity in development team *(Alpha One, Alpha Two, Beta One)*	Assign team full time to project *(Alpha One)* Leverage team diversity through cross team observation and close team member collaboration *(Alpha One)*
23. Inadequate planning and insufficient documentation (related to communicate face-to-face principle) *(Alpha Two, Beta Two)*	Support requisite exploration of alternative options *(Vidgen & Wang, 2009)* Provide opportunities to raise any concern for discussion in open forums *(Conboy & Morgan, 2011)*
24. Making decisions in development without consulting client (due to tight sprints schedules) *(Alpha Two)*	Engage team in evaluating agile opportunities and challenges *(Ramesh et al., 2010, McAvoy et al., 2012)*
25. Frequent change of IT representatives in client company *(Alpha One)*	Increase developers' business knowledge through client-organized training sessions *(Alpha One)* Build collaborative relationship with IT team in client organization *(Alpha One)*
26. Centralized decision making *(Vidgen & Wang, 2009)*	Expand project manager's role to include coaching and facilitation *(Conboy et al., 2010)*
Project Technology	
27. Lack of using high quality collaboration technologies and processes in development team *(Alpha Two, Beta One, Beta Two)*	Relocate developers to spend more time with each other *(Beta One, Beta Two)*
28. Lack of a good prototype to communicate requirements between stakeholders *(Alpha Two)*	Provide client with knowledge about agile projects through workshop *(Alpha Two)*
29. Employing agile methodology without planning up front *(Alpha Two, Beta Two)*	Discuss expectations and requirements with client and within team *(Beta Two)*
30. Prioritization of requirements based on one-dimensional thinking (related to working software principle) *(Augustine et al., 2005)*	Engage team in evaluating agile opportunities and challenges *(Ramesh et al., 2010, McAvoy et al., 2012)*
Project Setting	
31. Complex and domain specific project *(Alpha Two, Beta One, Beta Two)*	Discuss expectations and requirements with client and within team *(Alpha Two, Beta One)* Recruit experienced and motivated developers *(Beta Two)* Communicate importance of project to team members for key stakeholders and career opportunities *(Beta One)* Create and share goals within team *(Beta One)*
32. Small budget agile project with limited room for interaction *(Alpha Two)*	Engage team in evaluating agile opportunities and challenges *(Ramesh et al., 2010, McAvoy et al., 2012)* Improve team's agile and social skills through training *(Conboy et al., 2010)*
33. Dependence on existing or legacy technology *(Alpha One, Beta Two)*	Recruit experienced and motivated developers *(Beta Two)* Discuss expectations and requirements with client and within team *(Alpha One)*
34. Inability to choose development team members *(Beta Two)*	Help new team members integrate through mentors and incremental responsibilities *(Conboy et al., 2010)*
35. Different approaches to agility between development and client company *(Alpha One)*	Analyze client organization dynamics and be open to adapt to changing conditions *(Alpha One)*
36. Profit focused culture in development company *(Alpha Two)*	Put high value on mentoring and voluntary contributions in performance evaluations *(Conboy et al., 2010)*
37. Bureaucratic and centralized organizations *(Alpha One)*	Leverage positive relationships between client representatives and client management *(Alpha One)*

Table 2 affords a comparison between the high-performing and low-performing projects in terms of the risks they faced and the resolution actions they implemented. Table 3 summarizes the number of the risks and resolution actions in each project. As shown, the high performing projects (Alpha One, Beta One) did not necessarily experience fewer risks compared to low performing projects (Alpha Two, Beta Two) (column #2.Risk Items, Table 3). For example, Beta One faced 15 risks, which is more than risks that Beta Two faced (14 risks). Yet, high performing projects implemented more unique resolution actions compared to the risks they faced (column #3.Unique Resolution Actions-Risk Items, Table 3).

Table 3. Risk Items, Resolution Actions across Projects

Cases	# 1.Risk Items	#2.Unique Resolution Actions	#3.Unique Resolution Actions -Risk Items	#4.Total Resolution Actions	# 5.Total Resolution Actions - Risk Items
High-Performing Projects					
Alpha One	8	12	**4**	17	**9**
Beta One	15	7	**-8**	18	**3**
Low-Performing Projects					
Alpha Two	19	8	**-11**	13	**-6**
Beta Two	14	4	**-10**	11	**-3**

Table 2 (grounded understanding) indicates that all the four projects, at times, used the same resolution action to address more than one risk item. For example, Beta One applied ‘Create and share goals within team’ to address six different risk items. The comparison between high performing and low performing projects, in terms of implementing more unique resolution actions compared to the risks they faced, remains consistent when we pay attention to this repeated use. Specifically, Table 3 shows that Alpha One and Beta One (high-performing projects) applied 9 and 3 resolution actions more than the risks they faced, but Alpha Two and Beta Two (low-performing projects) implemented 6 and 3 resolutions actions less than the risks they faced (column #5.Total Resolution Actions - Risk Items, Table 3).

In addition, high performing projects, as compared to low performing ones, followed a bolder and more influential approach to risk resolution. For example, Beta One’s repeated use of ‘Communicate importance of project to team members’ and ‘Create and share goals within team’ was broadcasted to the team by the CEO and development manager at various formal and informal gatherings. Beta Two’s repeated use of ‘Recruit experienced and motivated developers’ as a dominant resolution action, however, mainly targeted resolving short-term project requirements. This created a sense of dependency to the expert members and an overall dissatisfaction (“Because we have [expert] people who have been here for such a long time we are absolutely dependent on them now.”)

Risk Management Model

Model Development

The section complements the findings of cross-case analysis (Table 2) with the vocabulary of risk-strategy analysis approaches (Iversen et al., 2004). The result is a risk management model (Figure 1) along with the categories of risk areas and resolution strategies (Table 4), and the associated risks and resolutions assessment frameworks (Tables 5-6).

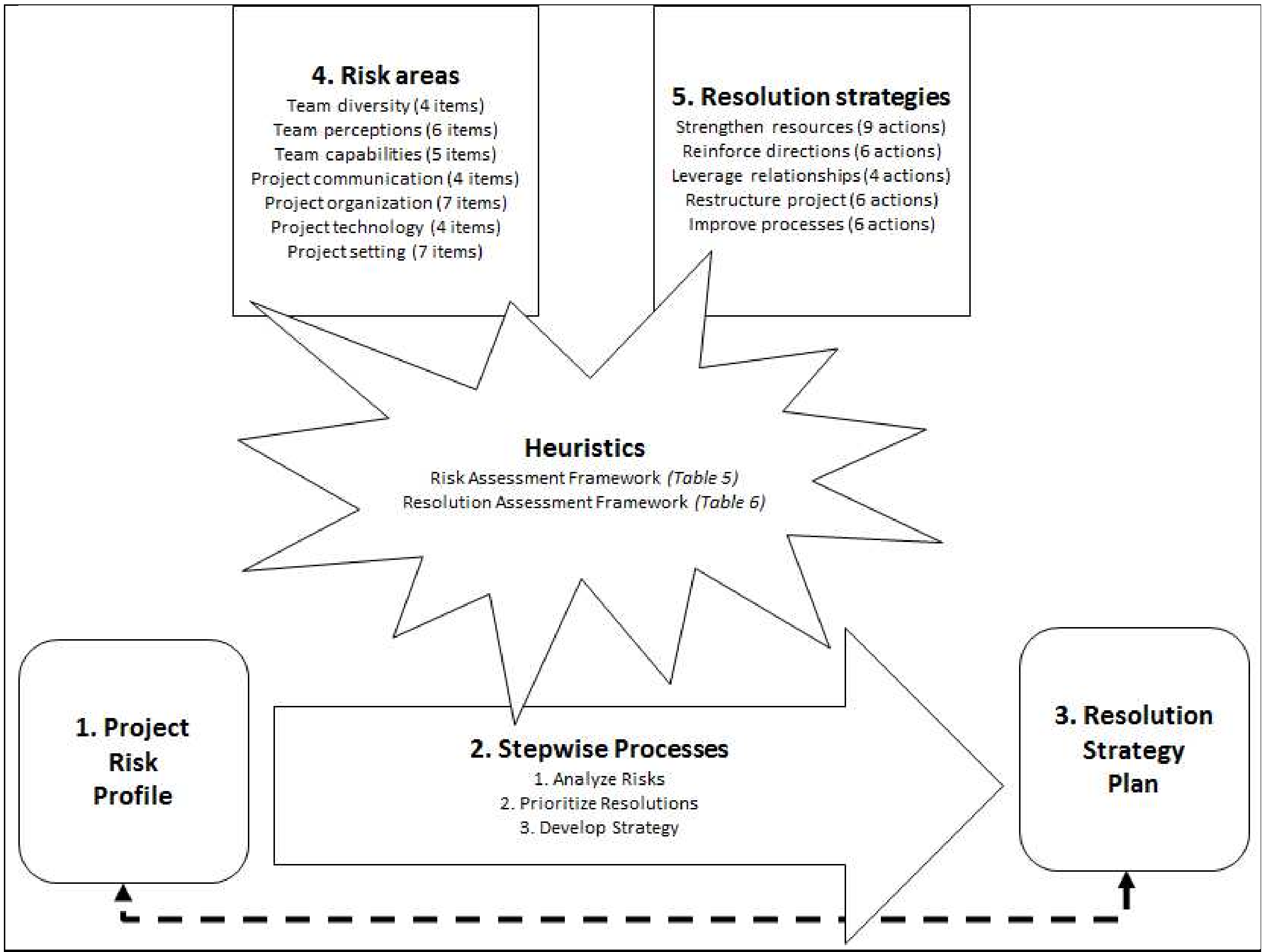
4. Risk areas
Team diversity (4 items)
Team perceptions (6 items)
Team capabilities (5 items)
Project communication (4 items)
Project organization (7 items)
Project technology (4 items)
Project setting (7 items)
5. Resolution strategies
Strengthen resources (9 actions)
Reinforce directions (6 actions)
Leverage relationships (4 actions)
Restructure project (6 actions)
Improve processes (6 actions)
Heuristics
Risk Assessment Framework (Table 5)
Resolution Assessment Framework (Table 6)
1. Project Risk Profile
2. Stepwise Processes
1. Analyze Risks
2. Prioritize Resolutions
3. Develop Strategy
3. Resolution Strategy Plan

Figure 1. Risk Management Model

Table 4. Conceptualization of Knowledge Sharing Risks and Resolutions

7 Risk Areas and 37 Risk Items	5 Resolution Strategies and 31 Resolution Actions
1. Team Diversity—refers to conceptual, geographical and time differences between team members that may pose risks to effective knowledge sharing 1.1. Different speaking languages among members 1.2. Different working and discipline-related backgrounds among members 1.3. Different time zones and physical distance between members 1.4. Lack of prior joint working experience in development team	**1. Strengthen Resources—refers to strategies that aim at developing supportive capabilities, experiences, and technologies** 1.1. Recruit developers with a combination of IT and business knowledge 1.2. Recruit experienced and motivated developers 1.3. Improve team's agile and social skills through training 1.4. Share historic and current systems documentation across team 1.5. Support client and team communication with collaborative

2. Team Capabilities—refers to knowledge and skill related issues amongst team members that may pose risks to effective knowledge sharing 2.1. Insufficient understanding of business domain and context 2.2. Unfamiliarity with development and collaboration technologies 2.3. Insufficient and ambiguous requirements 2.4. Inadequate social skills 2.5. Lack of familiarity with agile values and principles 2.6. Lack of IT resources and working experience with software companies in client company	technologies 1.6. Document experiences to support planning of future projects 1.7. Increase developers' business knowledge through client-organized training sessions 1.8. Provide client with knowledge about agile projects through workshops 1.9. Use pair programming to facilitate learning across development team
3. Team Perceptions—refers to attitudes and values of tram members that may pose risks to effective knowledge sharing 3.1. Lack of motivation, focus and adaptability in development team 3.2. Inappropriate assumptions about project scope made by client 3.3. Fear of self-exposure to technical and agile skills deficiencies in development team 3.4. Performance evaluation based on technical achievements 3.5. Stakeholder neglect of nonfunctional requirement	**2. Reinforce Directions—refers to strategies that aim at improving shared understanding of project goals and requirements within team and with client** 2.1. Communicate importance of project to team members for key stakeholders and career opportunities 2.2. Create and share goals within team 2.3. Discuss expectations and requirements with client and within team 2.4. Analyze client organization dynamics and be open to adapt to changing conditions 2.5. Collect and share successful project stories with team 2.6. Share key project information with client representatives using non-technical language
4. Project Communication—refers to communication-related issues within the project that may pose risks to effective knowledge sharing 4.1. Inadequate client availability and participation 4.2 Lack of communication of agile time requirements with client up front 4.3. Lack of concurrence within client team 4.4. Product owner lack of sharing client feedback with development team	**3. Leverage Relationships—refers to strategies that aim at improving working relationships within team and with client** 3.1. Leverage team diversity through cross-team observation and close team member collaboration 3.2. Promote positive relationships across stakeholders 3.3. Build collaborative relationship with IT team in client organization 3.4. Leverage positive relationships between client representatives and client management
5. Project Organization—refers to aspects of organization and conduct of the project that may pose risks to effective knowledge sharing 5.1. Tight sprints schedule with little time for interaction 5.2. Inadequate planning and organization in agile practices 5.3. Multitasking and lack of continuity in development team 5.4. Inadequate planning and insufficient documentation 5.5. Making decisions in development without consulting client 5.6. Frequent change of IT representatives in client company 5.7. Centralized decision making	**4. Restructure Project—refers to strategies that aim at restructuring resources to improve development team organization** 4.1. Delegate project responsibilities within team 4.2. Relocate developers to spend more time with each other 4.3. Help new team members integrate through mentors and incremental responsibilities 4.4. Change length of sprints to accommodate project's constraints 4.5. Assign team full-time to project 4.6. Expand project manager's role to include coaching and facilitation
6. Project Technology—refers to technological issues that may pose risks to effective knowledge sharing 6.1. Lack of using high quality collaboration technologies and processes in development team 6.2. Lack of a good prototype to communicate requirements between stakeholders 6.3. Employing agile methodology without planning up front 6.4. Prioritization of requirements based on one-dimensional thinking	**5. Improve Processes—refers to strategies that aim at improving development, communication, and evaluation processes** 5.1. Support participation and flexibility in project's decision making 5.2. Support requisite exploration of alternative options 5.3. Engage team in evaluating agile opportunities and challenges 5.4. Put high value on mentoring and voluntary contributions in performance evaluations 5.5. Provide opportunities to raise any concern inappropriate for discussion in open forums 5.6. Provide each individual member with 360° feedback
7. Project Setting—refers to task and context related issues that may pose risks to effective knowledge sharing 7.1. Complex and domain specific project 7.2. Small budget agile project with limited room for interaction 7.3. Dependence on existing or legacy technology 7.4. Inability to choose development team members 7.5. Different approaches to agility between development and client company 7.6. Profit focused culture in development company 7.7. Bureaucratic and centralized organizations	

Figure 1 demonstrates five components: (1) the project risk profile, (2) the stepwise process with heuristics, (3) the resolution strategy plan, (4) the risk areas and items, and (5) the resolution strategies and actions. The risk management model helps a software team move from a project risk profile to a resolution strategy plan. The project risk profile refers to existing or potential risks to effective knowledge sharing in the project. The resolution strategy plan refers to a plan of action with specified resolution strategies and their related resolution actions to mitigate the identified risks. The risk management process follows a stepwise process: (1) analyze risks, (2) prioritize resolutions, and (3) develop strategy plan. The process is supported by heuristics that link specific risks to appropriate resolutions. Heuristics help develop an overall resolution strategy plan for the project. The heuristics include the risk assessment framework (Table 5) and the resolution assessment framework (Table 6).

Application of the model begins with analyzing the project's risk profile (step 1, Figure 1) using the risk assessment framework (Table 5). The participants such as project managers, developers, and user representatives scan the 37 risk items. They circle relevant risk items (column Risk Area and Items, Table 5) and all the associated resolution actions (column Resolution Actions, Table 5). Utilizing a variety of techniques such as debating and voting across team members (Davis, 1982), they conduct a qualitative assessment (High/Medium/Low level of importance) of each risk item and risk area. They record the results of the assessment in the column Assess Risks (H/M/L), Table 5. For this assessment, team members should consider the probability and consequence of each risk item.

Table 5. Risk Assessment Framework

Risk Area and Items	**Assess Risks (H/M/L)**	**Resolution Actions**
1. Team Diversity		
◯ Different speaking languages among members		11
◯ Different working and disciple-related backgrounds among members		10, 11, 12, 20, 21
◯ Different time zones and physical distance between members		26
◯ Lack of prior joint working experience in development team		10, 11, 12, 20
2. Team Capabilities		
◯ Insufficient understanding of business domain and context		1, 2, 4, 7
◯ Unfamiliarity with development and collaboration technologies		9, 21
◯ Insufficient and ambiguous requirements		3
◯ Inadequate social skills		3
◯ Lack of familiarity with agile values and principles		27, 28
◯ Lack of IT resources and working experience with software companies in client company		8, 12, 13, 16, 17
3. Team Perceptions		
◯ Lack of motivation, focus and adaptability in development team		10, 11, 14, 21, 31
◯ Inappropriate assumptions about project scope made by client		12
◯ Fear of self-exposure to technical and agile skills deficiencies in development team		22, 27, 30
◯ Performance evaluation based on technical achievements		29

○ Stakeholder neglect of nonfunctional requirement		28
4. Project Communication		
○ Inadequate client availability and participation		13, 23
○ Lack of communication of agile time requirements with client up front		5, 12, 13, 15
○ Lack of concurrence within client team		1, 3
○ Product owner lack of sharing client feedback with development team		12
5. Project Organization		
○ Tight sprints schedule with little time for interaction		2, 10, 11
○ Inadequate planning and organization in agile practices		6
○ Multitasking and lack of continuity in development team		16, 24
○ Inadequate planning and insufficient documentation		27, 30
○ Making decisions in development without consulting client		28
○ Frequent change of IT representatives in client company		7, 18
○ Centralized decision making		20, 25
6. Project Technology		
○ Lack of using high quality collaboration technologies and processes in development team		21
○ Lack of a good prototype to communicate requirements between stakeholders		8
○ Employing agile methodology without planning up front		12
○ Prioritization of requirements based on one-dimensional thinking		28
7. Project Setting		
○ Complex and domain specific project		2, 10, 11, 12
○ Small budget agile project with limited room for interaction		3, 28
○ Dependence on existing or legacy technology		2, 12
○ Inability to choose development team members		22
○ Different approaches to agility between development and client company		13
○ Profit focused culture in development company		29
○ Bureaucratic and centralized organizations		19

Second, the team prioritizes the identified resolutions (step 2, Figure 1). The process begins by looking at the column Resolution Actions, Table 5. Team members count how many times each resolution action is circled for addressing risk items with (i) High, (ii) Medium, and (iii) Low levels of importance. They add these numbers to the column Add the Number of Targeted Risk Items, Table 6.

Table 6. Resolution Assessment Framework

Resolution Strategy	Resolution Action	Add the Number of Targeted Risk Items			Assess Resolution Action (H/M/L)	Assess Resolution Strategy (H/M/L)
		H	M	L		

1. Strengthen Resources	1. Recruit developers with a combination of IT and business knowledge					
	2. Recruit experienced and motivated developers					
	3. Improve team's agile and social skills through training					
	4. Share historic and current systems documentation across team					
	5. Support client and team communication with collaboration technologies					
	6. Document experiences to support planning of future projects					
	7. Increase developers' business knowledge through client-organized training sessions					
	8. Provide client with knowledge about agile projects through workshop					
	9. Use pair programming to facilitate learning across development team					
2. Reinforce directions	10. Communicate importance of project to team members for key stakeholders and career opportunities					
	11. Create and share goals within team					
	12. Discuss expectations and requirements with client and within team					
	13. Analyze client organization dynamics and be open to adapt to changing conditions					
	14. Collect and share successful project stories with team					
	15. Share key project information with client representatives using nontechnical language					
3. Leverage relationships	16. Leverage team diversity through cross team observation and close team member collaboration					
	17. Promote positive relationships across stakeholders					
	18. Build collaborative relationship with IT team in client organization					
	19. Leverage positive relationships between client representatives and client management					
4. Restructure project	20. Delegate project responsibilities within team					
	21. Relocate developers to spend more time with each other					
	22. Help new team members integrate through mentors and incremental responsibilities					
	23. Change the length of split sprints to improve interactions					
	24. Assign team full time to project					
	25. Expand project manager's role to include coaching and facilitation					
5. Improve processes	26. Support participation and flexibility in project's decision making					
	27. Support requisite exploration of alternative options					
	28. Engage team in evaluating agile opportunities and challenges					
	29. Put high value on mentoring and voluntary contributions in performance evaluations					
	30. Provide opportunities to raise any concern for discussion in open forums					

	31. Provide each individual member with 360° feedback					

Based on these numbers, the team qualitatively assesses each resolution action (column Assess Resolution Action (H/M/L), Table 6). Some guiding criteria could be the number of risk items that the resolution action can address, the number of highly-important risk items that the resolution action can address, and the feasibility of the resolution action. For example, it is possible that a resolution action can address important risks, yet it may not be a feasible action in that project. Final decision on whether it should be given high, medium, or low importance may include considering stakeholder interests and available resources, and it should involve techniques such as debating and voting. For example, the risk management process may suggest recruiting new developers as a highly important resolution action. However, due to project finances the team may decide to mitigate risks by increasing developers' business knowledge through client-organized training sessions (related to the category of Strengthen Resources actions). This process helps participants engage in mindfully understanding risks, potential resolution actions, challenges in executing them and alternative resolutions. The result of the assessment is recorded in the column Assess Resolution Action (H/M/L), Table 6.

Third, attention turns to development of an overall strategy to prioritize the five resolution strategies (step 3, Figure 1). For this, the team qualitatively assesses each of the five strategies (column Assess Resolution Strategy (H/M/L), Table 6). The team crafts a resolution strategy plan that includes: (i) prioritized resolution categories along with a list of their identified resolution actions, (ii) some notes on 'who, where and when' with regard to the resolution actions, and (iii) some notes on the role of key stakeholders or potential challenges in implementing the resolution actions. The main objective is to craft a bird eye's vision for the development team and to give them a head start on keeping the risk management direction simple and memorable (Olsen, 1988).

As a final note, identifying and mitigating risks should occur as early as possible. This is particularly the case in agile development. More specifically, the flexible characteristic of agile development adds additional change and customer-related concerns compared to traditional risk management frameworks. For example, the customer may express reluctance to continue with releases in small increments in the middle of the project (Lippert et al., 2003). Such characteristics suggest that agile development can particularly benefit from the risk management model. By engaging stakeholders in understanding the nature, consequences and management of risks, the team can constantly look for improvement opportunities. Agile practices such as sprint retrospective meetings can be leveraged for these purposes. At the end of each meeting, the team can quickly fill the forms (Tables 5-6). The team can keep the results for future use. For example, the forms can be scanned and stored in the company's knowledge management system. If this process was undertaken in prior meetings, the team can discuss: (i) the risks identified in previous meetings, (ii) the resolution actions that were implemented since then, and (iii) consequences of their implementation and any particular challenge or important issues that occurred in their implementation.

Model Evaluation

We conducted four one-hour evaluation sessions of the risk management model with two project managers, two developers, one tester and one user representative in one of the software companies studied. At the end of each session, the researcher asked the participants to provide additional feedback on the model's strengths and areas for improvement.

The participants agreed that the model is easy to use and it helps in the management of projects. The interviewees, especially project managers, expressed that the model offers a proactive approach to assess knowledge sharing risk items during projects and to identify resolution actions to mitigate the risks or prevent them from adversely affecting knowledge sharing practices. Notably, they asked for permission to keep the forms and apply the model within their company. For example, a project manager narrated that the model is not only useful in creating a shared vision and a prioritized plan, but it also helps reduce agile-related stress. He explained that he would like to include the risk management process model in regular retrospective action items: *"Scrum meetings can sometimes be stressful* ... The needs of the customers are unclear or changing ... There is a lot of work to be done, and team members have sometimes difficulties doing their work. This [model] creates a shared vision of priorities across the team. The [risk management] process is also fun. It has a different style that helps reduce work stress. I would put it on as a retrospective action item.*"* In another session, one of the senior developers argued that the model ensures stakeholders' concerns are considered thoroughly, planned for, and monitored: "The model gives the people who worry about the project a space to come in the meeting and put those worries down. We can plan for risks and monitor the results.*"* A tester expressed that it would be promising to build an application that automates and facilitates the risk management process. In contrast, project managers argued that the informal process of applying the model and its forms during retrospective meetings is an effective solution for their projects.

Discussion

Theoretical Contributions

This study was motivated based on a pressing theoretical and practical need to generate new insights on how agile teams may prevent communication-related and knowledge sharing barriers from adversely affecting agile development practices (Ramesh et al., 2010, Ghobadi & Mathiassen, 2015). We addressed the research question—how can agile development teams systematically assess and mitigate risks to effective knowledge sharing—by developing a risk management model. In an inductive, grounded fashion, we combined in-depth data collected from four software projects with theoretical insights from the agile literature (Conboy et al., 2010, Vidgen & Wang, 2009, McAvoy et al., 2012, Ramesh et al., 2010, Conboy & Morgan, 2011, Ghobadi & Mathiassen, 2015) and research on risk management approaches (Davis, 1982, Persson et al., 2009, Iversen et al., 2004). Our research presents three theoretical contributions.

First, the risk management model (Figure 1, Tables 4-6) adds substantive content to our understanding of communication-related and knowledge sharing issues in contemporary software teams that actively use agile practices. Specifically, the model contributes new concepts and detailed processes to develop a resolution strategy plan in response to a project's knowledge sharing risk profile. Such an empirically-grounded and theoretically-informed understanding has been absent from existing research and practice discourses. For example, software development research has identified risks and resolution actions for mitigating them (Keil et al., 1998, Ropponen & Lyytinen, 2000, Boehm, 1991), but there are scarce efforts that go beyond this foundational step to develop comprehensive risk management plans (Iversen et al., 2004). In contrast, our proposed approach provides heuristics that facilitate analyzing risks, prioritizing resolutions, and linking them into an overall plan. Our proposed risk management process also engages team members in several informal, mutual knowledge sharing exercises. These exercises help teams overcome decision making challenges that are common in agile contexts. Specifically, research suggests several challenges, such as lack of shared understanding and developers' lack of enthusiasm to communicate, may inhibit effective shared decision-making in agile teams (Moe et al., 2012). In response, a risk management process provides team members with an engaging opportunity to confront each other, discuss priorities (especially conflicting ones), and understand project complexities. The use of formalized methods may seem controversial in agile development research. Yet, rhetoric on agile and plan-driven approaches have become less confrontational in the past few years (Boehm & Turner, 2003). More recently, scholars have begun to observe real-world projects more closely, echoing the existence and importance of leveraging ambidexterity in agile teams (Ramesh et al., 2012, Ramesh et al., 2006). Our results and model evaluation findings concur with this view and add to it by showing that the risk management model is useful in many ways. Specifically, we showed that the model can easily be used during agile practices such as light-weight and informal scrum and retrospective review meetings, helping software teams achieve better performance and individual-related outcomes.

Second, our model defines and presents seven categories of risk areas and five resolution strategies. These conceptual categories are useful in studying and measuring several aspects of knowledge sharing in software development. For example, Table 4 outlines 4-9 'concepts' associated with each 'category'. Tables 5-6 offer heuristics for linking risks and resolutions. Thus, they refer to 'links' between concepts (risks and resolutions) and the link between conceptual categories (risk areas and resolution strategies). These conceptual categories, their associated concepts, and heuristics for linking concepts and categories lay the foundation for significant qualitative and quantitative investigations into communication-related challenges confronting agile teams.

Third, the empirical data reveals that the high performing projects did not always experience fewer risks compared to low performing projects, but they did implement more unique resolution actions compared to the risks they faced. Furthermore, although all projects used certain resolution actions for addressing different risks, high performing projects applied those actions in a bolder and more thoughtful manner compared to low performing ones. For example, high performing projects highlighted the organizational importance of the project and the managerial support in various occasions. These insights concur with prior software development research (Barki et al., 2001), suggesting that different project risk management profiles can lead to different project performance outcomes.

Practical Contributions

The risk management model meets the criteria of practical applicability proposed by (Glaser & Strauss, 1977). First, the model fits the substantive area of knowledge sharing in software development. Specifically, software teams constantly invest in better communication and knowledge sharing efforts (Gupta & Bajwa, 2012). They increasingly need intellectual tools that help identify and assess knowledge sharing risks and prevent them from adversely affecting development practices. In response to this practical need, we have offered a detailed risk management model (Figure 1, Tables 4-6).

Second, the model is sufficiently general to be relevant to a range of software development contexts. Specifically, it is grounded in empirical findings based on data collected from four projects in two software companies. In addition, its practical usefulness is strengthened using insights within the extant literature.

Third, our evaluation suggests the model is readily understandable by practitioners and provides useful guidance in the management of knowledge sharing practices in agile development. The model, therefore, serves as a basis on which software practitioners can iteratively assess risks to effective knowledge sharing and take important steps for mitigating them at different stages of development. In summary, the risk management model offers several practical advantages.

First, close and committed participation from stakeholders support collective mindfulness and team learning in agile development (McAvoy et al., 2012, Hoda et al., 2013, Keil et al., 2002). We recommend software teams involve different stakeholders in the risk management process. For example, they can conduct the process during project retrospectives. Second, the risk management process allows team members to discuss which actions did work or did not work over the project life-cycle. For example, the process may suggest 'providing each individual member with 360° feedback' to mitigate the risk that 'lack of motivation, focus and adaptability in development team' poses to effective knowledge sharing (Table 5-6). Project retrospectives may, however, suggest the following: This action proved to be effective at early stages, but providing feedback consumed the time of experienced developers, and slowed down their pace of work at later stages of development. Third, research suggests that reluctance to transmit bad news concerning a software project and its status can increase project losses (Tynjälä et al., 2009, Smith & Keil, 2003). In contrast, accumulated knowledge and lessons learned from regular risk management facilitates organizational learning, forms an open culture for transmitting both positive and challenging outcomes, and creates possibility of changing project direction for better. Fourth, regular risk management efforts help create and maintain a risk management registry for each specific project, enabling experience reuse and cross-project learning (Newell, 2004, Petter & Vaishnavi, 2008).

Concluding Remarks

We acknowledge a number of limitations that present opportunities for future theory development. First, the findings are limited to four agile projects across two software companies. We recommend large-scale empirical studies to validate, modify, or extend the presented model. Second, our model suggests a straightforward approach involving high, medium, low scores for assessing risks and resolutions. Shared-decision making for assessing risks can be challenging particularly in large projects that involve many stakeholders. Future research may develop and implement more systematic approaches for assessing risks. Examples include measuring the probability of risk multiplied by the loss associated with it, and measuring the magnitude of potential loss associated with project failure (Barki et al., 1993).

Third, we have evaluated the practical utility of the model in one software company. Research may provide additional insights by evaluating the model in different companies, at different stages of development (Tasharofi & Ramsin, 2007), and during different types of software projects (large, medium, small size). Scholars may apply design science techniques to develop web-based tools that support assessments of the model (Persson et al., 2009). Fourth, we recommend longitudinal research to extend the model by identifying the risks that may occur as the result of implementing certain resolution actions. For example, the resolution action 'analyze client organization dynamics and be open to adapt to changing conditions' can lead to major reworks when the architecture does not scale up (Batra, 2009). Thus, implementing this resolution action can make it difficult for developers to commit enough time to knowledge sharing activities at later stages of development. Fifth, longitudinal studies are encouraged to link resolution actions to project phases, advancing our understanding of different stages of development in which each resolution action is best implemented. For example, researchers may explore which resolution actions are best implemented prior to the project initiation. Sixth, our results suggest that the high performing projects, more than the low performing ones, tend to address risks more effectively by applying more resolution actions compared to the risks they face. Future research should further investigate differences between high performing and low performing projects in terms of their project's risk profile and their approach to the assessment and mitigation of knowledge sharing risks. Finally, we suggest sharing mental models is essential for successful team work (Ghobadi & Mathiassen, 2015). Yet, teams can also be subject to groupthink cognitive biases (Janis, 1982) that influence team members' understanding of risks and potential resolution actions. We recommend scholars study groupthink biases and their consequences in future risk management frameworks.

Acknowledgement

The authors thank the development teams who participated in this study as well as the senior editor, associate editor, and the three reviewers for their valuable guidance during the revision process.

References

Augustine, S., Payne, B., Sencindiver, F. & Woodcock, S. (2005) Agile project management: steering from the edges. Communications of the ACM, **48,** 85-89.

Barki, H. & Hartwick, J. (2001) Interpersonal conflict and its management in information system development. MIS Quarterly, **25,** 195-228.

Barki, H., Rivard, S. & Talbot, J. (1993) Toward an assessment of software development risk. Journal of Management Information Systems, **10,** 203-225.

Baskerville, R. L. & Stage, J. (1996) Controlling prototype development through risk analysis. MIS Quarterly, **20,** 481-504.

Batra, D. (2009) Modified agile practices for outsourced software projects. Communications of the ACM, **52,** 143-148.

Bellini, E., Canfora, G., García, F., Piattini, M. & Visaggio, C. A. (2005) Pair designing as practice for enforcing and diffusing design knowledge. Journal of Software Maintenance and Evolution: Research and Practice, **17,** 401-423.

Boehm, B. & Turner, R. (2003) Balancing agility and discipline: A guide for the perplexed, Addison-Wesley Professional.

Boehm, B. W. (1991) Software risk management: principles and practices. IEEE Software **8,** 32-41.

Carmel, E., Espinosa, J. A. & Dubinsky, Y. (2010) "Follow the Sun" Workflow in Global Software Development. Journal of Management Information Systems, **27,** 17-38.

Chakraborty, S. & Sarker, S. (2010) An exploration into the process of requirements elicitation: a grounded approach. Journal of the Association for Information Systems, **11,** 212-249.

Conboy, K., Coyle, S., Wang, X. & Pikkarainen, M. (2010) People over process: key people challenges in agile development. IEEE Software, **99,** 47-57.

Conboy, K. & Morgan, L. (2011) Beyond the customer: Opening the agile systems development process. Information and Software Technology, **53,** 535-542.

Corvera Charaf, M., Rosenkranz, C. & Holten, R. (2012) The emergence of shared understanding: applying functional pragmatics to study the requirements development process. Information Systems Journal, **23,** 115-135.

Dalcher, D. (2002) Safety, risk, and danger: A new dynamic perspective. Cutter IT Journal, **15,** 23-27.

Davis, G. B. (1982) Strategies for information requirements determination. IBM Systems Journal, **21,** 4-30.

Dingsøyr, T. & Hanssen, G. (2003) Extending agile methods: postmortem reviews as extended feedback, 4th International Workshop on Learning Software Organizations, Chicago, IL, USA: Springer Verlag, 4-12.

Dorairaj, S., Noble, J. & Malik, P. (2012) Knowledge Management in Distributed Agile Software Development, Agile Conference (AGILE), Dellas, Texas, US: IEEE, 64-73.

Dybå, T. & Dingsøyr, T. (2008) Empirical studies of agile software development: A systematic review. Information and Software Technology, **50,** 833-859.

Earl, M. 1987. Information Management Strategy. Englewood-Cliffs, NJ: Prentice-Hall.

Eisenhardt, K. M. (1989) Building theories from case study research. Academy of Management Review, **14,** 532-550.

Eisenhardt, K. M. & Graebner, M. E. (2007) Theory building from cases: opportunities and challenges. Academy of Management Journal, **50,** 25-32.

Fitzgerald, B., Russo, N. L. & O'Kane, T. (2000) An Empirical Study of System Development Method Tailoring in Practice, European Conference on Information Systems: Citeseer, 187-194.

Fitzgerald, B., Russo, N. L. & Stolterman, E. (2002) Information systems development: Methods in action.

Ghobadi, S. (2015) What drives knowledge sharing in software team: a review and classification framework. Information and Management, **52,** 82-97.

Ghobadi, S., Campbell, J. & Clegg, S. (2015) Pair programming teams and high-quality knowledge sharing: A comparative study of coopetitive reward structures. Information Systems Frontiers.

Ghobadi, S. & D'ambra, J. (2013) Modeling High-Quality Knowledge Sharing in Cross-Functional Software Development Teams Information Processing & Management, **49,** 138-157.

Ghobadi, S. & Mathiassen, L. (2015) Perceived Barriers to Effective Knowledge Sharing in Agile Software Teams. Information Systems Journal.

Glaser, B. G. & Strauss, A. L. (1977) The discovery of grounded theory: Strategies for qualitative research, Aldine Publishing, Chicago.

Gupta, N. & Bajwa, J. K. (2012) Analysis of Knowledge Sharing Practices in Distributed Agile Environment. International Journal of Computer & Communication Technology, **3,** 1-11.

Highsmith, J. (2009) Agile project management: creating innovative products, Pearson Education, MA, US.

Hoda, R., Babb, J. & Norbjerg, J. (2013) Toward Learning Teams. IEEE Software **30,** 95-98.

Holmström, H., Fitzgerald, B., Ågerfalk, P. J. & Conchúir, E. Ó. (2006) Agile practices reduce distance in global software development. Information Systems Management, **23,** 7-18.

Iversen, J. H., Mathiassen, L. & Nielsen, P. A. (2004) Managing risk in software process improvement: an action research approach. MIS Quarterly, **28,** 395-433.

Janis, I. L. (1982) Groupthink: Psychological studies of policy decisions and fiascoes, Houghton Mifflin, Boston.

Karlsen, J. T., Hagman, L. & Pedersen, T. (2011) Intra-project transfer of knowledge in information systems development firms. Journal of Systems and Information Technology, **13,** 66-80.

Keil, M., Cule, P. E., Lyytinen, K. & Schmidt, R. C. (1998) A framework for identifying software project risks. Communications of the ACM, **41,** 76-83.

Keil, M., Tiwana, A. & Bush, A. (2002) Reconciling user and project manager perceptions of IT project risk: a Delphi study. Information Systems Journal, **12,** 103-119.

Lee, A. S. & Baskerville, R. L. (2003) Generalizing generalizability in information systems research. Information Systems Research, **14,** 221-243.

Lippert, M., Becker-Pechau, P., Breitling, H., Roock, S., Schmolitzky, A., Wolf, H. & Heinz, Z. (2003) Developing complex projects using XP with extensions. Computer, **36,** 67-73.

Lyytinen, K. (1987) Different perspectives on information systems: problems and solutions. ACM Computing Surveys (CSUR), **19,** 5-46.

Lyytinen, K., Mathiassen, L. & Ropponen, J. (1998) Attention Shaping and Software Risk—A Categorical Analysis of Four Classical Risk Management Approaches. Information Systems Research, **9,** 233-255.

Mcavoy, J., Nagle, T. & Sammon, D. (2012) Using mindfulness to examine ISD agility. Information Systems Journal, **23,** 155-172.

Mcfarlan, F. W. (1981) Portfolio approach to information systems. Harvard Business Review, **59,** 142-150.

Moe, N. B., Aurum, A. & Dybå, T. (2012) Challenges of shared decision-making: A multiple case study of agile software development. Information and Software Technology, **54,** 853-865.

Nerur, S. & Balijepally, V. G. (2007) Theoretical reflections on agile development methodologies. Communications of the ACM, **50,** 79-83.

Newell, S. (2004) Enhancing cross-project learning. Engineering Management Journal, **16,** 12-20.

Newman, M. & Robey, D. (1992) A social process model of user-analyst relationships. MIS Quarterly, **16,** 249-266.

Olsen, R. J. (1988) Niche shock: And how to survive it. Planning Review, **16,** 6-13.

Persson, J. S., Mathiassen, L., Boeg, J., Madsen, T. S. & Steinson, F. (2009) Managing risks in distributed software projects: an integrative framework. IEEE Transactions on Engineering Management, **56,** 508-532.

Petter, S. & Vaishnavi, V. (2008) Facilitating experience reuse among software project managers. Information Sciences, **178,** 1783-1802.

Ramesh, B., Cao, L. & Baskerville, R. (2010) Agile requirements engineering practices and challenges: an empirical study. Information Systems Journal, **20,** 449-480.

Ramesh, B., Cao, L., Mohan, K. & Xu, P. (2006) Can distributed software development be agile? Communications of the ACM, **49,** 41-46.

Ramesh, B., Mohan, K. & Cao, L. (2012) Ambidexterity in Agile Distributed Development: An Empirical Investigation. Information System Research, **23,** 323-339.

Ropponen, J. & Lyytinen, K. (2000) Components of software development risk: How to address them? A project manager survey. IEEE Transactions on Software Engineering, **26,** 98-112.

Scott, W. A. (1955) Reliability of content analysis: The case of nominal scale coding. Public Opinion Quarterly, **19,** 321-325.

Smith, H. J. & Keil, M. (2003) The reluctance to report bad news on troubled software projects: a theoretical model. Information Systems Journal, **13,** 69-95.

Sutherland, J. (2005) Future of scrum: Parallel pipelining of sprints in complex projects, Agile Conference: IEEE, 90-99.

Tasharofi, S. & Ramsin, R. 2007. Process patterns for agile methodologies. In: Situational Method Engineering: Fundamentals and Experiences, pp. 222-237. Boston, Springer.

Tynjälä, P., Pirhonen, M., Vartiainen, T. & Helle, L. (2009) Educating IT project managers through project-based learning: A working-life perspective. Communications of the Association for Information Systems, **24,** 269-288.

Vidgen, R. & Wang, X. (2009) Coevolving systems and the organization of agile software development. Information Systems Research, **20,** 355-376.

Williams, L. (2012) What agile teams think of agile principles. Communications of the ACM, **55,** 71-76.

Appendix 1: Interview Guide

In this project, how important was ‘effective knowledge sharing” among team members?
In what ways was knowledge sharing practiced?
During the project, how satisfied were you by the knowledge sharing practices among team members?
In what ways did knowledge sharing help the project achieve its goals?
Which problems did you notice in achieving effective knowledge sharing among team members?
What were the key enablers of effective knowledge sharing?
What were the key barriers to effective knowledge sharing?
Now, I have noted a number of barriers to effective knowledge sharing in this project. Can you please have a look and sort them for me based on their level of importance.
Is there anything else that you would like to mention that we did not cover?
(Ghobadi & Mathiassen, 2015)

Appendix 2: Sample Coding

Codes	Sample Quote
High-performing project	*“Stakeholders got the results they wanted and they are happy with how they worked with the team.”*
Low-performing project	*“Project stakeholders didn’t see results fast enough.”*
Complex business rules	*“They [developers] knew what the algorithm was and what the code for it was, but understanding what the code’s ultimate goal was and spotting errors in the output was hard and time consuming” (user representative).*
Making experienced and motivated developers available	*“We had [a senior developer] who has the most experience of these products. We asked him how things work. If he hadn’t been available it would have been much more difficult”*
Lack of motivation in the team	*“I think they may have been less motivated, because they didn’t feel that their work was as important as what some of the other teams were doing”*
Emphasizing the organizational importance of the project	*“And, we are trying to basically let them be aware that they are in a position that is highly visible in the company and can grow very rapidly”*
Employing agile methodology without planning up front	*“I didn’t understand how much time would be required of me at the start.”*
Understanding client and being open and flexible to adapt to changing conditions	*“It seemed to me there was a clear awareness of the kinds of pressures that people in [my business] have. So when I had difficulty making myself available, they [development team] were very quick to adapt”*
Inadequate client availability and participation	*“Most of the people in [my business] have heavy workloads and administrative loads”*
Different working and discipline-related backgrounds among members	*“We had a huge amount of discrepancy in terms of people’s knowledge. We had someone who knows the [domain knowledge] over five years. We had people who had never worked on [this area].”*
Decreasing the length of sprints	*“Developers thought three weeks sprint to be better for their focus, because of rotating shifts with an alternative project.”*

Appendix 3: Project Summaries (Within-Case Analyses)

Project Alpha One Summary		
Risk Area	**Risks Item**	**Resolutions Actions**
Team Capabilities	Unfamiliarity with development and collaboration technologies	Use pair programming to facilitate learning across development team Relocate developers to spend more time with each other
	Lack of IT resources and working experience with software companies in client company	Discuss expectations and requirements with client and within team Analyze client organization dynamics and be open to adapt to changing conditions Leverage team diversity through cross team observation and close team member collaboration Promote positive relationships across stakeholders
Project Communication	Lack of communication of agile time requirements with client up front	Analyze client organization dynamics and be open to adapt to changing conditions Support client and team communication with collaboration technologies Share key project information with client representatives using nontechnical

		language Discuss expectations and requirements with client and within team
Project Organization	Multitasking and lack of continuity in development team	Leverage team diversity through cross team observation and close team member collaboration Assign team full time to project
	Frequent change of IT representatives in client company	Increase developers' business knowledge through client-organized training sessions Build collaborative relationship with IT team in client organization
Project Setting	Dependence on existing or legacy technology	Discuss expectations and requirements with client and within team
	Different approaches to agility between development and client company	Analyze client organization dynamics and be open to adapt to changing conditions
	Bureaucratic and centralized organizations	Leverage positive relationships between client representatives and client management

Project Alpha Two Summary

Risk Area	Risks Item	Resolutions Actions
Team Diversity	Different speaking languages among members	-
Team Perceptions	Lack of motivation, focus and adaptability in development team	-
	Inappropriate assumptions about project scope made by client	Discuss expectations and requirements with client and within team
Team Capabilities	Insufficient understanding of business domain and context	Share historic and current systems documentation across team Increase developers' business knowledge through client-organized training sessions
	Inadequate social skills	-
	Lack of IT resources and working experience with software companies in client company	Provide client with knowledge about agile projects through workshop Discuss expectations and requirements with client and within team Analyze client organization dynamics and be open to adapt to changing conditions
Project Communication	Inadequate client availability and participation	Change the length of split sprints to improve interactions Analyze client organization dynamics and be open to adapt to changing conditions
	Lack of communication of agile time requirements with client up front	Analyze client organization dynamics and be open to adapt to changing conditions
	Lack of concurrence within client team	-
Project Organization	Tight sprints schedule with little time for interaction	-
	Inadequate planning and organization in agile practices	-
	Multitasking and lack of continuity in development team	-
	Inadequate planning and insufficient documentation (communicate face-to-face principle)	-
Project Technology	Making decisions in development without consulting client	-
	Lack of a good prototype to communicate requirements between stakeholders	Provide client with knowledge about agile projects through workshop
	Employing agile methodology without planning up front	-

Project Setting	Complex and domain specific project	Discuss expectations and requirements with client and within team Recruit experienced and motivated developers Communicate importance of project to team members for key stakeholders and career opportunities
	Small budget agile project with limited room for interaction	-
	Profit focused culture in development company	-

Project Beta One Summary

Risk Area	Risk Item	Resolutions Actions
Team Diversity	Different speaking languages among members	Create and share goals within team
	Different working and disciple-related backgrounds among members	Communicate importance of project to team members for key stakeholders and career opportunities Create and share goals within team Delegate project responsibilities within team
	Different time zones and physical distance between members	Support participation and flexibility in project's decision making
	Lack of prior joint working experience in development team	Communicate importance of project to team members for key stakeholders and career opportunities Create and share goals within team Delegate project responsibilities within team
	Different speaking languages among members	-
Team Perceptions	Lack of motivation, focus and adaptability in development team	Communicate importance of project to team members for key stakeholders and career opportunities Create and share goals within team
	Fear of low estimates in development team	-
Team Capabilities	Insufficient understanding of business domain and context	-
Project Communication	Product owner lack of sharing client feedback with development team	Discuss expectations and requirements with client and within team
Project Organization	Tight sprints schedule with little time for interaction	Create and share goals within team Communicate importance of project to team members for key stakeholders and career opportunities Recruit experienced and motivated developers
	Inadequate planning and organization in agile practices	-
	Multitasking and lack of continuity in development team	-
Project Technology	Lack of using high quality collaboration technologies and processes in development team	Relocate developers to spend more time with each other
Project Setting	Complex and domain specific project	Discuss expectations and requirements with client and within team Communicate importance of project to team members for key stakeholders and career opportunities Create and share goals within team
	Multidimensional project involving both application and infrastructure development	-

Project Beta Two Summary

Risk Area	Risks Item	Resolutions Actions
Team Diversity	Different working and disciple-related backgrounds among members	Discuss expectations and requirements with client and within team Relocate developers to spend more time with each other
	Lack of prior joint working experience in	Discuss expectations and requirements with client and within team

	development team	
Team Perceptions	Lack of motivation, focus and adaptability in development team	Relocate developers to spend more time with each other
Team Capabilities	Insufficient understanding of business domain and context	Recruit experienced and motivated developers
	Unfamiliarity with development and collaboration technologies	-
	Insufficient and ambiguous requirements	-
Project Organization	Tight sprints schedule with little time for interaction	Recruit experienced and motivated developers
	Inadequate planning and organization in agile practices	Document experiences to support planning of future projects
	Inadequate planning and insufficient documentation (communicate face-to-face principle)	
Project Technology	Lack of using high quality collaboration technologies and processes in development team	Relocate developers to spend more time with each other
	Employing agile methodology without planning up front	Discuss expectations and requirements with client and within team
Project Setting	Complex and domain specific project	Recruit experienced and motivated developers
	Dependence on existing or legacy technology	Recruit experienced and motivated developers
	Inability to choose development team members	-

View publication stats