Technical Dependency Challenges in Large-Scale Agile Software Development

Nelson Sekitoleko[1], Felix Evbota[1], Eric Knauss[1], Anna Sandberg[2], Michel Chaudron[1], and Helena Holmström Olsson[3]

[1] Department of Computer Science and Engineering
Chalmers | University of Gothenburg
nellysek@gmail.com, gusevbfe@student.gu.se, eric.knauss@cse.gu.se
[2] Ericsson AB
[3] Malmö University

Abstract. This qualitative study investigates challenges associated with technical dependencies and their communication. Such challenges frequently occur when agile practices are scaled to large-scale software development. The use of thematic analysis on semi-structured interviews revealed five challenges: planning, task prioritization, knowledge sharing, code quality, and integration. More importantly, these challenges interact with one another and can lead to a domino effect or vicious circle. If an organization struggles with one challenge, it is likely that the other challenges become problematic as well. This situation can have a significant impact on process and product quality. Our recommendations focus on improving planning and knowledge sharing (with practices such as scrum-of-scrums, continuous integration, open space technology) to break the vicious circle, and to reestablish effective communication across teams, which will then enable large-scale companies to achieve the benefits of large-scale agility.

Key words: Technical dependencies, Large-scale agile, Cross-Functional Teams (XFT), Qualitative research

1 Introduction

Due to attractive characteristics such as flexibility, responsiveness and team empowerment, agile development methods have been increasingly adopted by large-scale development organizations. In emphasizing the use of iterations and development of small features, agile methods have increased the ability for software development companies to accommodate changing customer requirements and fast changing market needs [1]. In particular, agile methods have shown their capacity in empowering development teams, improving their relationship to customers, and allowing an increased focus on informal communication and coordination rather than focusing on formal communication and documentation of their practices [2]. As one of its basic principles, agile development provides simple, rapid, and incremental solutions to big problems by breaking down complex features into smaller ones [3]. This allows for small, cross-functional teams to work on smaller tasks and well-defined areas of development and hence, improve both efficiency and speed.

However, while agile development methods and the breakdown of complex tasks were originally developed to improve small-scale development, and to support co-located teams, they are being increasingly adopted by large-scale development organizations with globally distributed teams [3]. In such settings, the agile breakdown of complex tasks poses a big challenge due to complex technical dependencies between teams [4]. These dependencies can be seen in various ways, such as, dependencies between activities in the development process, dependencies among different software artefacts, and dependencies across teams and team members [5]. Looking back at the agile basic principles, teams should communicate directly in a face-to-face conversation [6], a situation that is rarely the case in large-scale distributed software development. Instead, the complexity of technical dependencies increases with the size of the company, and in large-scale software development the challenge therefore becomes how to minimize technical dependencies that have a negative impact on team performance, as well as how to enable communication and management of technical dependencies between teams.

Based on this identified challenge of technical dependencies in large-scale software development, this study addresses the following research questions:

RQ1: What are the challenges associated with technical dependencies between teams in a large-scale agile software development?
RQ2: What affects the likelihood of a challenge to occur?

The **contribution** of the paper is twofold. First, and based on case study research, we identify the challenges that exist in relation to technical dependencies between teams in large-scale distributed development. Second, we provide a set of recommendations on how software development companies can manage these challenges in order to mitigate the impact of these on development team performance. While our findings are based on a single case study, we believe that our findings are relevant to other organisations in which communication and coordination between teams is a critical task.

The remainder of this paper is structured as follows: Section 2 describes agile teams and how agile team practices are increasingly applied in large-scale software development. The section also introduces the notion of technical dependencies and how these can be communicated in teams and in between teams. Section 3 describes the research site and the case study research methodology that we applied in this study. In Section 4, we present the findings from the interview study. Finally, in Section 5 and 6 we discuss our findings as well as provide a set of recommendations to help organisations address the challenges we identified in relation to technical dependencies.

2 Background: Large-Scale Agility, Technical Dependency, and Communication

2.1 Agile teams

During the last decade agile methods have dramatically changed the way software development is performed, as well as the ways in which software teams are organised. Unlike traditional development methods characterized by plan-based execution of sequential phases, agile methods focus on managing unpredictability and change. In doing so, agile methods advocate small development teams in which all necessary competences are represented, i.e. cross-functional teams consisting of software developers, testers, architects etc. Typically, these teams take responsibility for the development of a software feature from the moment that a requirement comes from a customer, until that requirement is translated into software functionality that addresses that customers need. During development of a feature, the development team works in close collaboration with the customer in order to allow for rapid feedback loops, collaborative decision-making, as well as continuous integration and deployment of code changes [7]. In this way, agile teams seek to avoid cumbersome and time-consuming processes and instead focus on taking an end-to-end responsibility for feature development and that continuously validate if the functionality they develop correspond to customer needs. Typically, this is referred to as empowerment of teams [7]. Although agile methods differ in details and techniques, overall agile principles such as flexibility, empowered teams and customer collaboration lie at the heart of all of them.

2.2 Large-Scale Agile

For more than a decade, agile development methods have demonstrated their success in establishing flexible development processes with short feedback loops and consideration taken to evolving customer needs [2, 8]. Due to successful accounts [9, 10], these methods have become attractive to a broad variety of companies. Currently, large software-intensive organizations are in the process of deploying agile methods, and attempts to scale agile methods can be identified [11, 12, 13]. However, the applicability of these methods is not without challenges in large-scale development of software intended for a mass-market [14]. As recognized by Badampudi et al [15], organizations often discover misalignments between methods when attempting to use agile methods in a large-scale setting. According to the authors, the reason for this is that many large-scale companies practice agile in a way that is not consistent with the original agile ideas, and that the translation of the original ideas to a large-scale setting is difficult. Also, the shift towards agile is difficult for companies that are used to heavyweight sequential processes and companies that are confronted with interdependent teams and stakeholders located at different locations [15]. Often, development teams lack a shared understanding with other teams due to communication and coordination challenges, lack of documentation and complex decision-making processes among distributed stakeholders.

Another difficulty, and as reported on by Heikkila et al. [14], is the challenge related to cross-functional team creation. In their research, the authors identify difficulties with creating generalist teams that can implement features in all components of the software. As recognized in their study, organizations usually realizes that many components in a large-scale system are technically very difficult and interdependent, and require years of experience to be fully understood by developers. As a result, many large-scale organizations experience long lead times before the development teams can implement anything useful in a component. The authors conclude that identifying who has the required extensive experience and expertise to perform a task is still a challenge in a large organization adopting agile methods.

Finally, creating user stories that can be developed in a single sprint is reported on as challenging because of the complex nature of large-scale software systems [14]. Often, internal and external dependencies affect the way in which agile practices can be applied in large-scale development settings, and many organizations experience inconsistencies with the way in which agile practices are adopted. As recognized by [16], the understanding of the contingencies, i.e. how and when agile practices are applicable under variations in project size, business domain, and team configurations surrounding large-scale agile development is important.

2.3 Technical dependency

The large number of interdependencies among activities and artefacts in the software development process is one of the major challenges in large-scale software development, which includes a large number of developers and development teams.[17]. Babinet and Ramanathan identify the following challenges of technical dependencies [18]:

- Unpredictability, were teams find it difficult to know beforehand what changes, issues, surprises, failures and successes they will come across during the development of a feature.
- Conflicting priorities, such as a team depending on a component that has lower priority in the backlog of another team.
- Difficulty in understanding overlapping and short release cycles, and teams constant changing of priority in each sprint.

To address these technical dependencies, Babinet and Ramanathan recommend release kick-offs, dependency identification exercises, Scrum-of-Scrums, virtual architecture teams, status reports and a number of other activities that help team communication and knowledge sharing [18]. In similar, Souza et al. propose tools with which technical dependencies can be analyzed and visualized so that these are better understood and therefore, easier to communicate among development teams [19].

2.4 Communication

Communication is often described as fundamental for organizational success [20]. Effective internal and external communication stimulates the performance of a development organization [21]. However, while communication is central to all organizations it also poses major challenges. As recognized by Johansson et al., a message can be properly communicated but the intended receiver may choose to interpret the message as invalid [22]. Also, to select a message at one point and deliver that message at another point is problematic [23]. As experienced in most organizations, inter-team communication is a challenge that grows with the size and complexity of the organization.

3 Research Method

3.1 Research Setting

The case study was conducted at Ericsson AB. Ericsson provides communications networks, telecom services, and support solutions used in global communication. It is ranked the fifth largest software supplier in the world with 950 million subscribers in over 180 countries. In this section, we map and describe the concepts of *cross-functional teams (XFT)* and *technical dependencies* from an Ericsson perspective.

Cross-functional teams. A cross-functional team (XFT) is a team which has all core competences needed for the development and release of a feature. At Ericsson AB, XFTs generally follow the same working practices and include roles like system manager, system designer, function tester, system testers, and architect. In addition, each XFT has a scrum master, agile coach, and an operative product owner (OPO) on a part time basis and the team works in an open space which facilitates easy communication among teams. Each of these XFTs consists of 5–9 team members who have up to three roles in their team and some team members are associated with several teams in different roles. XFTs do not have team leaders but should be self-organized and work together with other XFTs on features which have a life cycle of approximately 500–1000 hours (a release consists of 20–80 features). Features are broken down into work packages which are developed in sprints of $\sim$3 weeks. During the sprint, a XFT takes full responsibility for the development of a work package, breaks it down into user stories and tasks, and is in charge of handling planned and unplanned technical dependencies. Our study focusses on the 30 XFTs responsible for the development of one specific embedded software product that has been developed during a period of more than 10 years with a design base of more than 1 Million lines of code.

Technical Dependencies. At Ericsson, technical dependencies are relationships and interactions between artifacts and teams during product development.

Examples include situations when a developer/team needs information regarding technical aspects of a system developed by another developer/team in order to progress the development work. Technical dependencies can occur during design-time, compile-time, and run-time and affect areas like source-code, architecture, hardware, and tools. At Ericsson, there are two types of technical dependencies: *Planned technical dependencies* are identified during the planning phase. Managers, program officers and product owners are responsible for identifying and scheduling planned technical dependencies, i.e identifying the tasks to be done in parallel or in sequence across teams, and communicating them to teams before development begins. *Unplanned technical dependencies* occur unexpectedly during the actual development of a product, for example due to improper implementation of the original plan.

3.2 Research approach

This paper reports on a three-months case study at Ericsson AB plus a follow-up questionnaire[1] three month later. A qualitative research approach was chosen to investigate our research questions from a social, technical and organizational context. As qualitative research approaches aim to investigate and improve the understanding of phenomena in their real-life context [24], and especially when the purpose is to explore peoples' experiences and perceptions, we found it particularly well suited for our interests.

3.3 Data collection

We interviewed 9 employees at Ericsson AB who were selected qualitatively based on insider knowledge about skills, experience, and organizational distribution from a population of 300 software engineers [25]. Our 9 interviewees have an average working experience of 10 years and about 3 years in agile practices, since Ericsson is gradually, team by team, transitioning to agile. In their work, they follow the most common agile practices like sprint-demo, retrospectives, daily stand-up meetings ($\sim$15min), and backlog grooming. Table 1 shows the roles the interviewees hold. A semi-structured interview approach was used to collect data because it has inherent properties that allow the interviewer to improvise and explore interview questions further [24]. Thus based on the progression of the interview, questions can be adapted and relevant follow-up questions posed [24]. The interview guide[2] helped us in ensuring that all questions were covered irrespective of the order in which they were followed. The interview questions mainly focused on planned and unplanned technical dependencies faced by XFT teams. Some of the interview questions are about the impact of incompatible components, how technical dependencies are located, communicated, resolved, and so forth [26]. We recorded the conversations while interviewing and transcribed these voice recordings verbatim to reduce the risk of corrupt data which can happen when transcribing during the interview [27].

[1] https://dl.dropboxusercontent.com/u/13255493/Tech-Depend-Questionnaire.pdf
[2] https://dl.dropboxusercontent.com/u/13255493/Tech-Depend-Interview-guide.pdf

Table 1. Interviewees and their roles / responsibilities.

ID	Role	Responsibility
P1	Software designer	SW development
P2	Software designer and scrum master	SW development, facilitate team work
P3	Function tester	Functional testing
P4	Software designer	SW development
P5	Software designer and scrum master	SW development, facilitate team work
P6	Scrum master and architect	Team support, technical leadership
P7	Software designer and scrum master	SW development, facilitate team work
P8	Function tester	functional testing
P9	System manager, scrum master, and Function tester	Give directions, facilitate team work functional testing

After analysis of the interview data, we collected additional data to be able to confirm our findings based on a questionnaire that presented our findings as statements in a short questionnaire with a likert scale to measure the agreement of the initial interviewees with challenges, their dependencies, and proposed solutions.

3.4 Data analysis

We analyzed the data collected from interviews base on the thematic analysis approach [27], an accepted method with wide-spread use in scientific and social science research consisting of six phases [27]. Please refer to [26] for details and examples of the data analysis.

1. Familiarizing with the data: We transcribed and read the data from the 9 interviews.
2. Generating initial codes: We coded the data from the perspective of the research questions.
3. Searching for themes: We grouped the initial codes we generated into different groups that we refer to as *initial themes*.
4. Reviewing Themes: We reviewed the initial themes, regrouped and refined them by cross checking the interview data with the generated codes in Phase 1 and 2. We then extracted and refined 5 themes in Phase 3.
5. Defining and naming themes: In this phase we reached a consensus about the five themes, which, in accordance to our research questions, we named the *main challenges* and present in the results section.
6. Producing the report: In this paper, we present and discuss the five main challenges and make recommendations.

3.5 Threats to Validity

As recognized by Maxwell [28], qualitative researchers rarely have the benefit of previously planned comparisons, sampling strategies, or statistical manipulations that control for possible threats to validity. While this can be acheived in quantitive research, qualitative researchers must try to rule out validity threats after the research has begun by using evidence collected during the research itself to make alternative hypotheses or interpretations implausible. One important aspect of validity is construct validity [24] that reflects to what extent the operational measures that are studied represent what the researcher has in mind, and what is reflected in the interview questions and themes. To address this critical aspect, we started each of our interviews with an introduction part in which the researchers shared their understanding of agile practices and technical dependencies with the interviewee. For example, we shared different definitions of the agile concept and we discussed the agile manifesto to get a shared understanding for the values that underpin agile development methods. Also, we shared our understanding of technical dependencies and why it is important to consider these in a large-scale development setting. In this way, the researchers and the interviewee had a shared understanding of the topic before the interview started, and we could proceed with asking questions without having to worry about the interviewee being unsure about the context we studied. With respect to external validity, i.e. to what extent it is possible to generalize the findings, our contribution is related to (1) the drawing of specific implications and (2) the contribution of rich insight [29]. Based on our interview findings, we present implications in a particular domain of action, i.e. in a particular software development company. Our study brings together empirical insight that allows for a deep understanding of this particular company, and the findings we present should be regarded as insights valuable for other companies interested in understanding the impact of technical dependencies in large-scale agile development.

4 Analysis and Interpretation

4.1 Technical Dependency Challenges in Large-Scale Agile

With respect to RQ1, (*What are the challenges associated with technical dependencies between teams in a large-scale agile software development?*), the analysis of the interview data revealed five main challenges: the planning challenge, the task prioritization challenge, the knowledge sharing challenge, the code quality challenge, and the integration challenge.

Planning challenge. A perfect plan for software development would minimize the occurrence of technical dependencies. Uncertainty, which is inherent to software development, is one reason why creating and following such an optimal plan is practically impossible, but our interviewees also indicate potential for realistic improvement. This is reflected by the following quote from one of our interviewees:

> *"[Managers] do not plan and allocate tasks to teams in an appropriate way because they do not know much about the code and do not involve in the actual coding."*

Our interviewees mentioned that sometimes tasks that should have been assigned to a single team were instead split and assigned to several teams, thereby creating unnecessary dependencies. Insufficient planning leads to unplanned technical dependencies during the actual product development. Such unplanned technical dependencies across teams do not occur frequently, but when they occur, they seriously impact development and lead to changes in requirements and time-plan. Our interviewees also said that it is difficult to locate the root cause of unplanned technical dependencies.

Task prioritization challenge. According to our interviewees, the *task prioritization challenge* a result of the *planning challenge*. When unplanned technical dependencies arise, teams have to update their sprint plan to account for changes in requirements and time-plans. These changes arise for example from new requests for components from other teams that were not planned before and often lead to conflicts in the product backlog. Two problematic scenarios given by our interviewees characterize the prioritization challenge:

(1) When teams have to implement a component which was not in their backlog and (2) when they have to deliver a component in their backlog earlier than scheduled since another team realized that they were dependent on the component.

According to our interviewees, the above scenarios led to re-prioritizing tasks in their backlog.

> *"[...] constant changing of priorities makes our burn-down charts look bad."*

Our interviewees stated that changing priorities in their backlog usually destabilizes their work plan, because they need to assign resources to the unplanned requests, thereby leading to delays and late deliveries.

Knowledge sharing challenge. From the perspective of our interviewees, knowledge sharing among the XFTs is vital to enable good communication and coordination. If knowledge is not properly circulated, communicating technical dependencies will suffer, as indicated by some of the problems raised by the interviewees.

- Some interviewees do not have the opportunity to say what they want in company meetings (e.g. tasks presentation meetings), because of the multitude of people in the meeting. The interviewees claimed they do not get opportunity to express their burning issues or raise vital questions.
- Experienced personnel is involved in difficult tasks and often too busy to be approached.

Our interviewees also expressed some concerns about some of their colleagues attitude and ability to share knowledge, including the following problems:

- Protectiveness: Some team members are protective of their work and do not want to provide support to others.

- Bad teachers: Some team members know much about the code, but are simply not good at explaining it.
- Laziness: Some team members do not want to share knowledge because they fear that others will start seeking help from them more often.
- Over specialization: Some team members prefer to focus on their own task, thereby not having adequate knowledge of the entire product, which in turn leads to inefficient communication about dependencies.
- Lack of communicativeness: Some team members are too shy to either ask or provide information during meetings, thus causing important information to be ignored.

Another problem related to knowledge sharing occurs when team members do not understand, ignore, or forget what was discussed in a meeting:

> *"During development some people forget easily what was agreed upon in scrum meetings. Then, they are not be able to work accordingly."*

From the perspective of our interviewees it is clear that such problems with knowledge sharing create a major challenge for communicating technical dependencies.

Code quality challenge. In software companies, good code quality will lead to quality products that can compete favourably in the market. However, in large scale software development, maintaining good quality code remains a challenge. Our interviewees stated that despite the existence of Subversion (SVN) control tools, too many people involved in the same code make changes in the code which can end up as conflicts in other teams. Their common view was:

> *"Such changes make it difficult to maintain a stable version of code, hence reducing code quality and creating more technical dependencies."*

Function testers specifically shared an opinion that such changes make testing more complex because they have to rewrite test cases many times. The prevailing view among our interviewees was that providing good quality code is difficult because of technical dependencies.

Integration challenge. In large-scale agile software development, merging of work packages is a problem because of the many self-organized teams working to deliver an integrated working product to the customers. Our interviewees demonstrated a scenario in which teams develop work packages independently for 2-3 months without knowing what is happening in the main branch. At delivery, teams get conflicts since many changes have been made in the main branch, hence creating dependencies which at times may only be resolved by engaging other teams. Despite tool support, this is a challenging task.

Other concerns expressed by interviewees were about incompatible dependent components they received from other teams that resulted in merge conflicts. According to our interviewees, incompatible components often cause teams to either re-plan or re-develop their work, thereby leading to late deliveries. It appears that the integration challenge is a result of not handling technical dependencies in a good way.

4.2 Likelihood of Technical Dependency Challenges

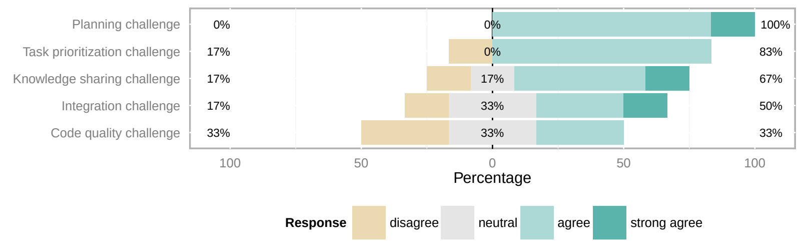

Fig. 1. Agreement of interviewees with challenges (”It is challenging to . . . ”).

With respect to our RQ2 (*What affects the likelihood of a challenge to occur?*), we first tried to achieve a better understanding of the nature of the challenges we identified. By doing this, we then found that in fact the likelihood of a challenge to occur is affected by the presence of other challenges, for example if the planning challenge is not resolved, then it can lead to other challenges.

Understanding the nature of challenges. For understanding the nature of the challenges we discovered, we asked our interviewees which challenges they consider to be most dominant in their daily work. Fig. 1 shows that planning and task prioritization are recognized as most challenging.

During discussion and analysis of our findings, we also recognized that some of the challenges we found are more technical in nature (code quality and integration challenge), while others can be characterized as communication challenges (see axis Fig. 2). In fact, the *knowledge sharing* challenge refers to the mindset of engineers, which is mostly related to communication. *Task prioritization* and *planning* refer to work practices and relate both to communication and technical challenges. *Code quality* and *integration* are mostly technical in nature and require technical actions.

Relationships and interdependencies. Based on the improved understanding of the challenges, a critical study of the main challenges by the authors revealed that during the development of a product, the challenges interact with one another to form a domino effect which leads to the technical dependency loop (Fig. 2).

These relationships between challenges cause a vicious circle. Consider for example the planning challenge: By suboptimal planning, unnecessary technical dependencies are introduced. These cause problems that surface as task prioritization challenge, which in turn increase the integration and code quality challenges. Bad code quality can put additional pressure on teams which are then reluctant to share knowledge. This in turn makes planning even more difficult. These circular relationships are bidirectional, e.g. in the example above, unresolved prioritization issues in the teams' backlogs seriously impair adequate planning.

4.3 Recommendations

In order to break through the vicious circle, one has to start with mitigating one challenge and then continue to exploit the positive influence on other challenges. Our interview data suggests that the knowledge sharing challenge is a good starting point. Improved knowledge sharing between technical and management staff on different levels can significantly improve the ability to create a good plan, and then in turn help addressing the other challenges.

We were especially interested in how our interviewees rate the dependencies and were specifically asking, whether a solution for the planning (respectively:

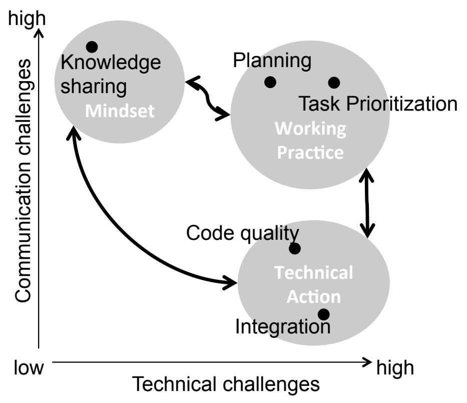

Fig. 2. Challenges associated with technical dependencies can be classified as communication and technical challenges. These challenges affect each other.

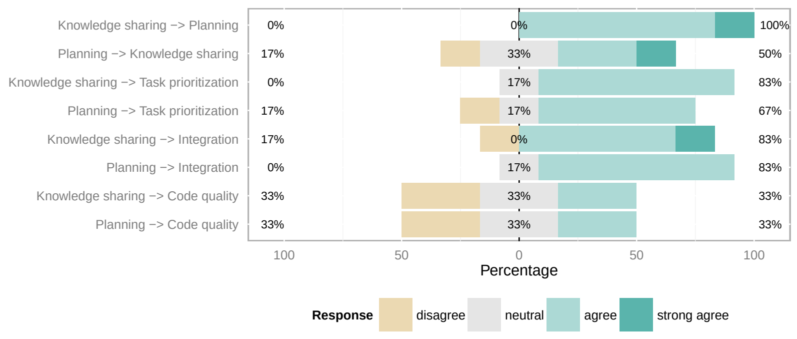

Fig. 3. Rating of challenge dependencies by interviewees.

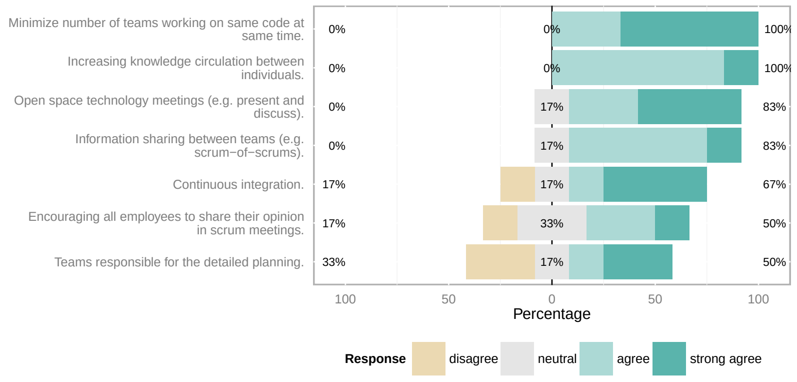

Fig. 4. Agreement of participants with recommendations.

knowledge sharing) challenge would positively impact other challenges as well. Fig. 3 indicates that a solution for the knowledge sharing challenge would have more impact on the other challenges. The figure also shows that the code quality challenge is a bit detached from the vicious circle, as our participants do not agree that a solution for another challenge would positively impact the code quality.

In order to gain a richer understanding of the knowledge sharing and planning challenge, we presented a number of recommendations from literature for mitigating the challenges to our interviewees. Fig. 4 shows the agreement of our interviewees with the recommendations we made.

5 Discussion

5.1 Implications for Practitioners

It is not new for practice that technical dependencies are cumbersome in large software development. When scaling agile, the technical dependencies do not become more or less, they just become more obvious and this is actually a possibility for practice to deal with them. Here we have come to understand challenges associated to technical dependencies and the domino effect this can create. By embracing this knowledge of the domino effect, practice can break the vicious circle by improving one or two of the challenges and by that improving all challenges. For example, when improving the Planning challenge by making sure the planners have sufficient competence of the code, the Task prioritization challenge get less problematic. Planners need to understand the quality of the impacted code to make correct estimations (e.g. stinker code is known to take ten times more time than code included in lean components), which then helps prioritize task in the right order. For practice, it is of high importance to understand not only the challenges on detailed level, but also how they impact each other in order to improve where it gets the most impact. This study gives practice such understanding.

5.2 Implications for Research

Eklund and Bosch propose a model for defining interactions necessary for agile teams together with a set measures facilitating agile development in a context where the full product cannot be agile [30]. These interactions can be seen in four categories; requirements, project gates, integration & validation, and delivery. The more teams understand such interactions, the less technical dependencies (as discussed in this paper) they will encounter. In contrast, unplanned technical dependencies can surface in late and inefficient clarification of features and requirements, as discussed in related work on patterns of continuous requirements clarification [31].

Cataldo et al. show that lack of socio-technical congruence, i.e. the fact that social relationships such as communication of developers is not aligned with technical dependencies between them, leads to bad software quality [32]. Damian et al. discuss similar observations from a case study where organizational structure is not in line with the partitioning of requirements, thus leading to unsatisfied communication needs [33]. Both works hint on potential for organizational optimization, which however is especially difficult in volatile, complex, and large-scale agile environments.

Promising avenues for future research therefore include (i) investigating ways to measure interaction and knowledge sharing quality and providing actionable feedback to agile teams and (ii) gain a better understanding how organizational change can support minimizing technical dependencies.

6 Conclusion and Outlook

In this qualitative study, we identified five challenges associated with technical dependencies in large-scale agile software development: planning, task prioritization, knowledge sharing, code quality, and integration. More importantly, we found that these challenges interact and can lead to a domino effect or vicious circle: If an organization struggles with one challenge, it is likely that the other challenges become problematic as well. A follow-up questionnaire confirmed the relationships between challenges as well as that mitigating one of the challenges can have a positive impact on the other challenges and ultimately promises to break the vicious circle. Our results indicate that activities should focus on mitigating the knowledge sharing and planning challenges to reestablish effective communication across teams, which will then enable companies to achieve the benefits of large-scale agility. Although the findings in this paper are based on a single case study, we believe that our findings are relevant to other companies transitioning towards large-scale agile development practices.

References

1. Fogelström, N.D., Gorschek, T., Svahnberg, M., Olsson, P.: The impact of agile principles on market-driven software product development. Journal of Software Maintenance and Evolution: Research and Practice **22** (2010) 53–80
2. Highsmith, J., Cockburn, A.: Agile software development: The business of innovation. IEEE Computer **34**(9) (2001) 120–122
3. Kettunen, P., Laanti, M.: Combining agile software projects and large-scale organizational agility. Softw. Process **13** (2008) 183–193
4. Beck, K.: Embracing change with extreme programming. Computer **32**(10) (1999) 7077
5. Curtis, B., Krasner, H., Iscoe, N.: A field study of the software design process for large systems. Commun. ACM **31** (1988) 1268–1287
6. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D.: Manifesto for agile software development. http://www.agilemanifesto.org (accessed on Dec-3, 2013) (2001)
7. Highsmith, J.: The great methodologies debate: Part 2. Cutter IT Journal **5** (2002)
8. Larman, C., Vodde, B.: Scaling Lean and Agile Development: Thinking and Organizational Tools for Large-Scale Scrum. Pearson Education Inc., Boston, MA (2009)
9. Abrahamsson, P., Warsta, J., Siponen, M., Ronkainen, J.: New directions on agile methods: a comparative analysis. In: Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon (2003) 244–254
10. Olsson, H.H., Alahyari, H., Bosch, J.: Climbing the stairway to heaven: A multiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In: Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications, Cesme, Izmir, Turkey (2012)
11. Kerievsky, J.: Industrial xp: Making xp work in large organizations. Executive report in Agile Project Management **6**(2) (2005)
12. McMahon, P.E.: Extending agile methods: A distributed project and organizational improvement perspective. In: Proceedings of the 17th Annual Systems and Software Technology Conference, Salt Lake City, UT (2005)
13. Lagerberg, L., Skude, T., Emanuelsson, P., Sandahl, K., Stahl, D.: The impact of agile principles and practices on large-scale software development projects: A multiple-case study of two projects at ericsson. In: ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement, Baltimore, Maryland (2013) 348–356
14. Heikkila, V.T., Paasivaara, M., Lassenius, C.: Scrumbut, but does it matter? a mixed-method study of the planning process of a multi-team scrum organization. In: ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement, Baltimore, Maryland (2013) 85–94
15. Badampudi, D., Fricker, S., Moreno, A.: Perspectives on productivity and delays in large-scale agile projects. In Baumeister, H., Weber, B., eds.: 14th Int'l Conf. on Agile Processes in Software Engineering and Extreme Programming (XP '13). Volume 149 of LNBIP., Vienna, Austria, Springer (2013) 180–194
16. Dingsøyr, T., Moe, N.B.: Research challenges in large-scale agile software development. SIGSOFT Softw. Eng. Notes **38**(5) (August 2013) 38–39

17. de Souza, C.R.B., Redmiles, D.F., Mark, J., Penix, G., Sierhuis, M.: Management of interdependencies in collaborative software development. In: Proc. of Intl. Symp. on Empirical Software Engineering. (2003) 294–302
18. Babinet, E., Ramanathan, R.: Dependency management in a large agile environment. In: Proc. of Agile Conference. (2008) 401–406
19. de Souza, C.R.B., Quirk, S., Trainer, E., Redmiles, D.F.: Supporting collaborative software development through the visualization of socio-technical dependencies. In: Proceedings of the 2007 International ACM Conference on Supporting Group Work, Sanibel Island, Florida, USA (2007) 174–154
20. Dainton, M., Zelley, E.D.: Applying communication theory for professional life: a practical introduction. SAGE Publications Inc. (2005)
21. Sosa, M.E., Eppinger, S. D. Pich, M., McKendrick, D.G., Stout, S.K.: Factors that influence technical communication in distributed product development: an empirical study in the telecommunications industry. IEEE Transactions on Engineering Management **49** (2002) 45–58
22. Johansson, B.J.E., Persson, P.A.: Reduced uncertainty through human communication in complex environments. Cogn. Technol. Work **11** (2009) 205–214
23. Shannon, C.E., Weaver, W.: The Mathematical Theory of Communication. University of Illinois Press (1971)
24. Runeson, P., Hst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Engg. **14** (2009) 131–154
25. Creswell, J.W.: Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. SAGE Publications (2009)
26. Sekitoleko, N., Evbota, F.: Technical dependencies in practicing agile in large-scale software development organizations: A case study conducted at Ericsson AB. Bachelor thesis, University of Gothenburg, Sweden (2013) https://dl.dropboxusercontent.com/u/13255493/Tech-Depen-Report.pdf.
27. Braun, V., Clarke, V.: Using thematic analysis in psychology. Qualitative Research in Psychology **3** (2006) 86–94
28. Maxwell, J.: Qualitative research design: An interactive approach. Sage, Los Angeles (2013)
29. Walsham, G.: Interpretive case studies in is research: nature and method. European Journal of Information Systems **4** (1995) 74–81
30. Eklund, U., Bosch, J.: Applying agile development in mass-produced embedded systems. In: Proceedings of 13th International Conference on Agile Software Development, Malmö, Sweden (2012) 31–46
31. Knauss, E., Damian, D., Poo-Caamao, G., Cleland-Huang, J.: Detecting and Classifying Patterns of Requirements Clarifications. In: Proceedings of 20th International Requirements Engineering Conference (RE '12), Chicago, USA (2012) 251–260
32. Cataldo, M., Herbsleb, J.D., Carley, K.M.: Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In: Proceedings of Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM'08). ESEM '08, Kaiserslautern, Germany, ACM (2008) 2–11
33. Damian, D., Helms, R., Kwan, I., Marczak, S., Koelewijn., B.: The role of domain knowledge and hierarchical control structures in socio-technical coordination. In: Proc. of IEEE Int. Conf. on Software Engineering (ICSE), San Francisco (2013)