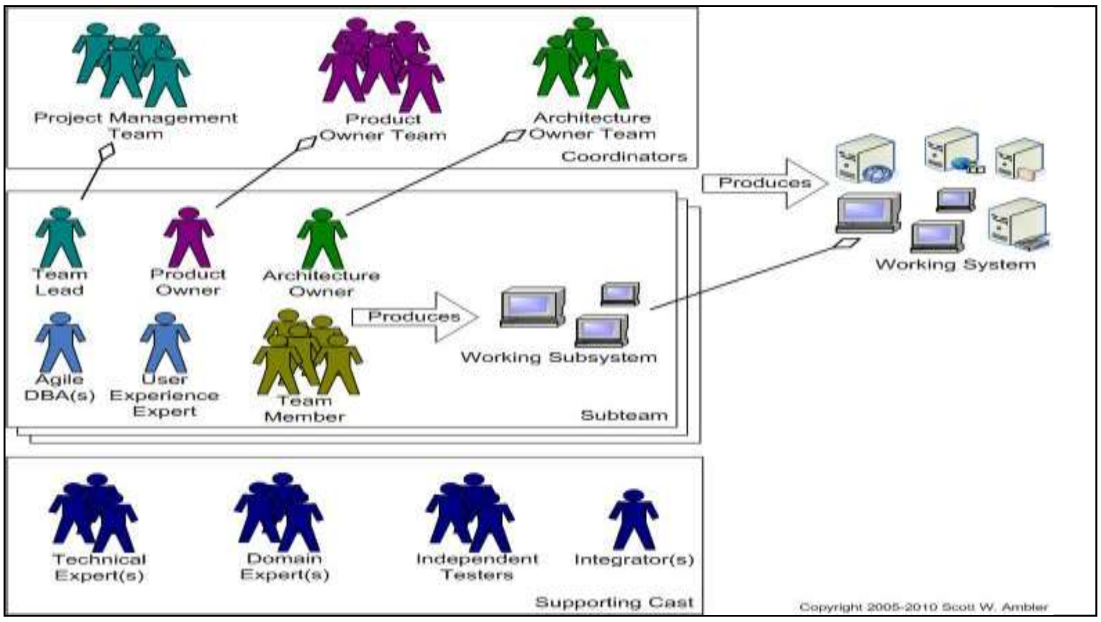
UNIVERSITY OF GOTHENBURG



Technical dependencies in practicing Agile in large-scale Software Development Organizations

A case study conducted at Ericsson AB

Bachelor of Science Thesis in Software Engineering and Mangement

Nelson Sekitoleko
Felix Evbota

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, June 2013

Technical dependencies in practicing Agile in large-scale Software Development Organizations
A case study conducted at Ericsson AB

Nelson Sekitoleko
Felix Evbota

Examiner: Ana Magazinius

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover picture taken from: http://www.ambysoft.com/essays/agileRoles.html

Department of Computer Science and Engineering
Göteborg, Sweden June 2013

Technical dependencies in practicing Agile in large-scale Software Development Organizations

A case study conducted at Ericsson AB

Nelson Sekitoleko
Dept. of computer Science and Engineering
University of Gothenburg, Sweden
nellysek@gmail.com

Felix Evbota
Dept. of Computer Science and Engineering
University of Gothenburg, Sweden
gusevbfe@student.gu.se

Abstract

Due to the benefits associated with Agile practices, such as flexibility, responsiveness. Large-scale software companies have been attracted to scale Agile practices which has led to software vices like technical dependencies. This study investigates the challenges associated with technical dependencies, and the challenges of communicating technical dependencies in large-scale Agile software development. A qualitative research approach was used to investigate the study. Thematic analysis of the interview data revealed: Planning, Teams backlog priority, Attitude and Knowledge sharing, Code quality, and Merge challenges, as the main challenges of this study. The main challenges interact with each other forming a technical dependency loop, and lead to domino effect, during the development of a product. The magnitude of the domino effect will determine the quality of the final product. We suggested some recommendations such as broadening initiative, continuous integration among others, to mitigate the above challenges. Resolving the challenges of technical dependencies will lead to effective communication across teams, which will enable large scale companies realize the benefits of large scale agility.

Keywords

Technical dependencies, Agile, Cross-Functional Teams (XFT)

1 INTRODUCTION

1.1 Problem definition

Agile practices provide simple, rapid, and incremental solutions to big problems by breaking down complex features into smaller ones. These smaller features are developed across small, flexible, co-located, or globally distributed software teams. Such Agile setting poses a big challenge of technical dependencies, and communication across teams. Technical dependencies can be seen in various ways, such as, dependencies among activities in the development process, dependencies among different software artifacts, for example source-code dependencies across teams [3]. The Agile manifesto recognizes that despite the availability of processes and tools, teams should communicate directly in a face-to-face conversation [15]. However, the complexity of technical dependencies increase with the size of the company, which leads to breakdowns in communication across large-scale Agile teams [3]. Such communication breakdowns leave the original assumption of face-to-face communication a locked principle in the Agile manifesto [7]. It is worth noting that minimizing software technical dependencies facilitates software understanding, reuse, and testing [12]. It is on this basis that we investigate how large-scale Agile teams manage, and communicate technical dependencies.

This study **aims** to identify and address the challenges of technical dependencies across large-scale Agile software development to enable them communicate effectively during software development. Effective communication will enable large-scale companies realize the benefits of large scale agility, such as mass production, global presence, and outsourcing [16].

The study addresses the following research questions:
RQ1.What are the challenges associated with technical dependencies across teams in a large-scale Agile software development?
RQ2. What are the challenges of communicating technical dependencies across teams in large-scale Agile software development?

This study contributes to Software Process Improvement (SPI) literature.
We exclude technical issues such as developing an application to solve an organizational problem. The study focuses on Ericsson AB as our case study setting.
This paper is structured as follows: Section 2 describes Ericsson case, Section 3 describes related work. Section 4 describes the methodology. In Section 5 we present the results from the interview study. In section 6 we discuss the results, and then conclude the paper with recommendations.

2 Ericsson case

For the purpose of this research, in the next subsections, we present Ericsson facts, the definition of technical dependencies used in this paper, Cross-Functional Teams (XFT), and task break down at Ericsson.

2.1 Ericsson AB facts

Ericsson AB provides communications networks, telecoms 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.

2.2 Definition Technical Dependencies

At Ericsson **technical dependencies** are artifacts interactions developers encounter within their teams or while working with other teams. They exist when a developer/team needs information regarding technical aspects of a system from another developer/team in order to progress in his or her development work.
Technical dependencies usually occur during design, compile-time, and run-time. Teams usually have dependencies in areas like source-code, architecture, hardware and tools.

At Ericsson the most common types of technical dependencies are:
Planned technical dependencies: these are identified during the planning phase. They involve identification of tasks to be done in parallel or in sequence across teams, and they are explicitly explained to teams before development begins.
Unplanned Technical dependencies: these are dependencies that occur by surprise during the actual development of a product. They may occur due to failure to implement the original plan.

2.3 Cross-Functional Teams.

XFT is a team which has all core competences needed for the development of a feature from product planning to product release. At Ericsson AB an XFT comprises of roles like, system manager, system designer, function tester, system testers, and architect. In Addition, each XFT on a part time basis has a Scrum-Master, Agile coach, and an Operative Product Owner (OPO). XFTs teams do not have team leaders with an ambition of making them self-organized and empowered over time. This means teams take full responsibility for the development of their work package and they are in charge of handling planned and unplanned technical dependencies.

2.4 Task breakdown

At Ericsson AB, a pre-study of tasks is done, which involves, task breakdowns, and prioritization, technical dependencies identification exercises, among others. During the planning phase the planned technical dependencies are identified, they are presented before tasks are assigned to the teams for actual development.

3 RELATED WORK

3.1 Large Scale agility

Agile methodologies have been primarily recommended to small, self-organizing, collocated teams, having ready access to interactive customers with a view of closing the communication gap between the business community and the developers [7], [15]-[18]. Does this mean that large-scale software companies that do not share these Agile paradigms are denied of Agile benefits? [16]. Leffingwell [16] recommends that large companies should learn from the original Agile practices and try to apply Agile practices to large-scale software development. Ericsson AB is one the companies that have applied Agile methodologies on large-scale through their developed process called Streamline Development (SD) [18], [19]. Two Other studies that have been conducted on large-scale agility are described below:

Kettunen and Laanti [7] investigated how and when agility could be utilized in large-scale software product development. They proposed the agility framework which involves organizations understanding the: (i) goals of agility, for example, productivity, (ii) means of agility, for example, software platforms, and (iii) enablers of agility, for example, human factors, for guiding Software Process Improvement (SPI) in large-scale software organizations. They recommend companies to have a holistic system wide view of software agility in order to improve software development.
Leffingwell [16] describes seven Agile team practices that natively scale to large organizations:

1. The define/build/test (d/b/t) component team
2. Two level planning and tracking
3. Mastering the iterations
4. Smaller, and more frequent releases
5. Concurrent testing
6. Continuous integration
7. Regular reflection and adaptation

However despite all the efforts by practitioners to scale software agility, new vices like technical dependencies are still potential threats to large-scale software agility. In the next section, we draw our focus on understanding how other researchers and practitioners manage technical dependencies in large-scale agility.

3.2 Technical dependencies

One of the reasons why cooperative software development is challenging is because of the large number of interdependencies, such as interdependencies among activities in the software development process, interdependencies among different software artifacts, and interdependencies in different parts of the same artifacts [4]. The research conducted by Babinet and Ramanathan [1] shows that unpredictability is one of the biggest challenge of technical dependencies across teams. They stated that teams find it difficult to know beforehand what changes, issues, surprises, failures and successes they will come across during the development of a feature. In addition to that, Babinet and Ramanathan also saw conflicting priorities, such as a team depending on a component that has lower priority in the backlog of another team, as another challenge of technical dependencies across teams. Babinet and Ramanathan pointed out more challenges, such as system complexity, difficulty in understanding overlapping and short release cycles, team constant changing of priority in each sprint.

Research shows that some of the ways of addressing technical dependencies are release kickoff, dependency identification exercise, release open space, Scrum-of-Scrums (SoS), Virtual Architecture Team (VAT), status report, functional design reviews, and Continuous Integration [1]. Souza et al. [12] and Trainer et al. [20] see Ariadne as approach of addressing technical dependencies. They stated that Ariadne is a plug-in for Eclipse, and that Ariadne is used for analyzing software projects for dependencies, and collects authorship information about projects relying on configuration management repositories. Ariadne can translate technical dependencies among components such as source-code modules into social dependencies among developers [12], [20].)Researchers have also adopted an approach of creating mechanisms in programming languages to minimize dependencies between software elements [12].

Parnas [9] points out that information hiding is the most important approach in minimizing dependencies, because information hiding motivates several mechanisms in programming languages, including data encapsulation, interfaces, and polymorphism. Information hiding uses the concept of coupling and design patterns which gives run-time program dependencies explicit representation as static program structures, making the dependencies easier to address [9], [21]. The field of Software Engineering has also developed tools like configuration management system and issues-tracking systems to overcome the problem of technical dependencies [12].

3.3 Communicating technical dependencies

Communication is an underlying principle that guarantees organizational success [13]. Internal and external communication that is effective stimulates the performance of a development organization [30]. Dainton and Zelley [13] state that there is no guarantee that organizations will be successful if they acquire a particular set of skills because most organizations have self-contradictory idea about communication. The basic problem of communication is to select a message at one point and deliver the exact message in another point [22].

Johansson and Persson [6] state that there is a challenge of uncertainty in communicating technical dependencies. They emphasized that when there is communication between humans; individuals only get the last grasp of the message that is communicated to them [6]. Furthermore, Johansson and Persson state that there is a deception in communication between individuals, a message can be properly communicated but the intended receiver may choose not to accept the message as valid.

De Souza et al. [4] state that there is a challenge of limitation with formal approaches, such as software development process, division of labour, formal meetings, software engineering tools like configuration management systems, bug-tracking tools, and so forth that large-scale Agile teams adopt to communicate technical dependencies. The same challenge of limitation is also associated with informal approaches, such as conventions, partial check-ins, problem reports (PRs), and e-mails in communicating technical dependencies [4].

4 METHODOLOGY

4.1 Research approach

We conducted the study using a qualitative research approach [2], [10]. This being a case study about understanding the challenges of technical dependencies, and communicating technical dependencies across teams at Ericsson, qualitative research approach was suitable because it was designed to help researchers understand social and institutional context from participants point of view [2], [10].

4.2 Data collection

We interviewed 9 employees at Ericsson AB, who were selected using a convenience sample [2]. We chose convenience sample because it was not easy to gain access to the employees, hence we only focused on those interviewees that were available. Thus, we maintained ethical standards because the interviewees consented to participate in the study [10]. The 9 interviewees were representatives of 30 XFT teams of 5-9 developers developing the same huge complex product. Some interviewees play one or more roles. We have masked the names of the interviewees in accordance with Singer and Vison [11], and Curtis et al. [3] to maintain confidentiality about their identity.

We used a semi-structured interview approach to collect data because it allows for improvisation and exploration [10], we asked the interview questions based on the development of the conversation between us and the interviewees. The interview guide 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 (see appendix 1). We used a voice recorder to record the conversations while interviewing, which we later played to carry out a verbatim transcription of the recorded interviews. Transcribing after conducting the interviews reduces the risk of having corrupt data, unlike direct transcription during interview which increases the risk of corrupt data [14].

Table 1: Interviewees and their roles

Participants	**Roles**
A	Software designer(a.k.a Programmer)
B	System designer and Scrum master
C	Function Tester
D	Software designer
E	Software designer and scrum master
F	Scrum Master and Architect
G	Software designer and scrum master
H	Function tester
I	System manager, Scrum Master and Function Tester

4.3 Data analysis

We analyzed the data collected from interviews using thematic analysis approach [14]. We opted for this approach because it is a well-known method used in scientific and social science research with six phases which are easy to apply [14].

Phase1: Familiarizing ourselves with the data
We transcribed and read the data from the 9 interviews.
Phase2: Generating initial codes
We coded the data from the perspective of the research questions [14]. The table below shows an extract of the code generated from particular part of the interview transcript.

Table 2: Sample of interview transcript and generated codes

Sample of interview transcript	Generated codes
Yeah it is too little technical people involved in assigning out the tasks to different teams (8.01), that is the problem, because if only managers and project managers do this, they don't know much about the code, I think maybe we want more technical people then we can be better.........	Too little technical people involve in giving out tasks

Phase3: Searching for themes
We grouped all the initial codes we generated into different groups that we referred to as initial themes or challenges of our research questions. To see all initial themes, see **appendix 3**.

Phase 4: Reviewing Themes:
We reviewed the initial themes, regrouped and refined them by cross checking the interview data and the generated codes in phase 1 and phase 2. Five main themes were extracted and refined from initial themes in phase 3.

Table 3: sample table representing how we grouped the codes to generate five main themes.

Codes	Challenge/theme
• Some people prefer to focus on their own task thereby not having product general picture which lead to inefficient communication and dependencies • People who are so protective of their work and end up saying that is your problem • Some team members do not want to share knowledge and tools due to fear of providing support. ……….	Attitude and knowledge sharing challenge

Phase 5: Defining and naming themes.
A consensus was reached about the five themes. Which we named the main challenges of our research questions presented in the results section.
Phase 6: Producing the report:
We presented, and discussed the five main challenges, and made recommendations.

5 RESULTS

The analysis of the interview data revealed **five main challenges** associated with technical dependencies, and communicating technical dependencies, across large-scale Agile software development namely:
C1. Planning challenge
C2. Teams backlog priority challenge
C3. Attitude and knowledge sharing challenge
C4. Code quality challenge
C5. Merge challenge

A further analysis of the five main challenges revealed that they can be grouped into **three categories: working practices, mindset, and technical action**. The challenges were grouped depending on when they occur, and the impact they have on development of the product.

Figure 2 illustrates a **visual representation** of the main challenges and categories as those which are: **technical challenges** that arise as a result of teams depending on different software artifacts such as code and **communication challenges** that arise as result of the way teams communicate technical dependencies

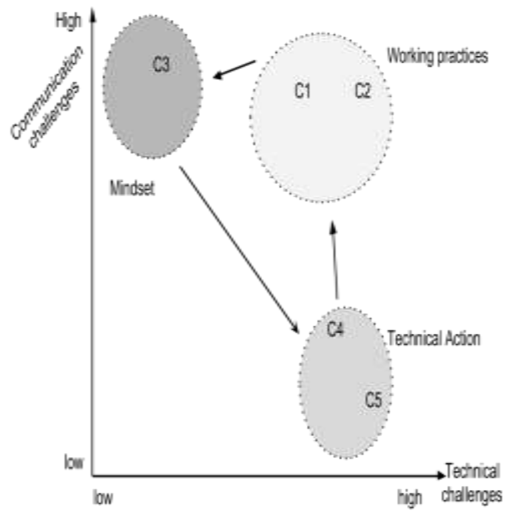


Figure 2 visual representation of the main challenges and categories

Example from **Fig 2**, C3-Attitude and knowledge sharing challenge is high towards the communication challenges axis and low on technical challenges axis, which implies that C3 is strongly a communication challenge.

Detailed explanation of the main challenges and the categories they form

The main challenges and categories are elaborated explicitly using actual statements of the interviewees on how they manage technical dependencies.

I. Working practices

These are challenges that relate to the way of working in the organization. For example, how tasks are divided and prioritized. The challenges in this category include:

Planning challenge

From the perspective of our interviewees, the ability to plan and predict the future minimizes the occurrence of technical dependencies during product development. However, coming up with plan that correctly predicts the future, and implementing that plan across teams still remains a challenge in software development. This planning challenge is reflected in our interviewee's view who stated that, managers do not plan and allocate tasks to teams in the appropriate way because they do not know much about the code. Our interviewees mentioned that there are instances where a task that is supposed to be assigned to a single team, is instead split and assigned to several teams, thereby creating unnecessary dependencies that would have been avoided.

Failure to have the right plan will lead to unplanned technical dependencies during the actual product development. Our interview data revealed that unplanned technical dependencies are minimal across teams but when they occur, they lead to changes in requirements and time-plan. They also said that it is difficult to locate the exact source of unplanned technical dependencies.

Team backlog priority challenge

This challenge arises as a result of planning issues. When unplanned technical dependencies arise teams have to try to update the new changes into their current plan. These changes arise from the new requests for components from other teams that were not planned before. These unplanned requests lead to conflicts in the product backlog. Our interviewees gave two scenarios when they were requested: (1)
To implement a component which was not in their backlog and (2) to deliver a component in their backlog earlier than planned since another team realized that they were dependent on the component.

According to our interviewees, the above scenarios led to re-prioritizing of tasks in their backlog. 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. Other interviewees said that constant changing of priorities usually make their burn-down charts look bad.

II. Mindset:

These are challenges that relate to the way individuals or teams perceive and respond to issues that arise during the development of a product. For example, when unplanned technical dependencies arise, what attitude does a team member shows towards resolving or communicating the technical dependencies to other teams. **In this category we have:**

Attitude and knowledge sharing challenge

In large-scale Agile software development, knowledge sharing among the XFTs is vital to enable the XFTs have a good communication and coordination. If knowledge is not properly circulated, it will lead to a challenge of communicating technical dependencies. Our interviewees stated that:

- Some interviewees do not have the opportunity to say what they want in company meetings, for example, 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.
- The experienced personnel that are involved in difficult are too busy to be approached.

Our interviewees also expressed concern about some of their colleagues' attitude towards knowledge sharing. Their opinions are presented below:

- People who are so protective of their work, and do not want to provide support to others.
- The people that know much about the code, but are not good at explaining when someone ask for help
- People that do not want to share knowledge and tools because of the idea that people will keep seeking help from them.
- People that prefer to focus on their own task thereby not having adequate knowledge of

the entire product, that usually lead to inefficient communication and dependencies

- People who are shy that usually do not understand when they communicate in meetings. They claimed that people who are shy to talk might have ideas that would have enhanced the knowledge of others.

During development some people forget easily what was agreed upon in scrum meeting, thereby not be able to work in accordance with what was agreed on. Some of our interviewees claimed it is a challenge with knowledge sharing, since those people did not absorb what was discussed in meeting. From the perspective of our interviewees it is clear that attitude and knowledge sharing is a challenge of technical dependencies and communicating technical dependencies.

III. **Technical action**

These are challenges relating to technical issues that require technical resolutions. For example, when team A delivers an incompatible component to the main branch and it causes many conflicts. Team A is advised to re-develop another component.

C4: **Code quality challenge**

In software companies, products are mainly defined by the lines of codes written and fully tested before release. Therefore good code quality will lead to quality products that can compete favorably on 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 the 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.

C5: **Merge 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. Some respondents pointed out that resolving merging conflicts is not so problematic but the tools they use like IBM Rhapsody makes code merging unnecessary difficult. Other interviewees were concerned with the difficulties in identifying the source of the conflict.

Other concerns expressed by interviewees were about incompatible dependent components they ordered from other teams that resulted in merge conflicts. This leads to project re-planning or re-developing that usually leads to late deliveries. The views of the interviewees above provide evidence that merge conflict is a result of technical dependencies.

6 DISCUSSION

In this section the main findings of the study are discussed, and compared to the findings of other researchers about the challenges of technical dependencies and challenges of communicating technical dependencies in large-scale Agile software development.

A further analysis of the main challenges and the categories reveals that they interact with one another to form a **technical dependency loop,** and lead to a **domino effect** during the development of a product. It is worth noting that this conclusion was reached by the authors of this paper after a critical analysis of the challenges and categories

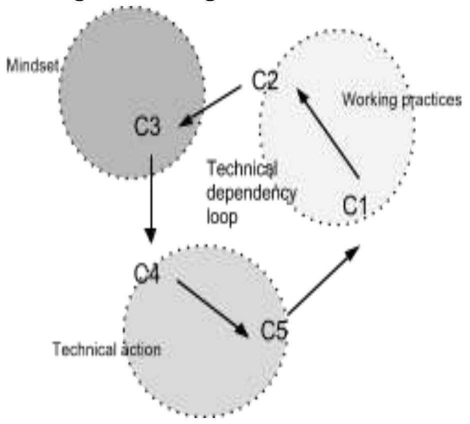


Figure 3: Illustration of the Technical dependency loop

The magnitude at which the challenges reinforce and impede [24] each other will determine the quality of the final product. For example, the working practice of an organization has influence on the type of mindset individuals in the organization have, and this type of mindset will greatly influence the quality of products in the organization.

Explanation of the technical dependency loop and the Domino effect

During the planning phase if close attention is not paid to the way tasks are divided and prioritized, there are high chances of identifying only few technical dependencies. When these tasks are deployed to the teams, during development unplanned technical dependencies will begin to arise, which indicates that there were **planning difficulties** (C1). This planning challenge is similar to what Babinet and Ramanathan [1] saw in their research. They used the term unpredictability. They claimed that the biggest challenge of technical dependencies is unpredictability. They emphasized that it is "impossible to anticipate all the issues, surprises, changes, failures and successes that teams will encounter during software development

Teams will struggle to fit these unplanned technical dependencies into their current product backlog, thereby leading to **conflicts in their backlogs (C2).** Conflicting priorities is a situation whereby a certain team depends on a feature that has a lower priority in another team's backlog [1]. Interviews data showed Unplanned technical dependencies destabilize teams' plan, lead to time wasting, delays and late deliveries. Ciborra [26] and Mathiassen [27] refer to such planning conflicts as drifting forces that drift technology away from original plans. Similarly Boehm and Turner [25] refer to such conflicts as organizational antibodies, which are also similar to what is described in chaos stage in the Satir-Change Model [28].

At this stage new plans are inserted into the product backlog due to the unplanned technical dependencies that have arisen. This may create a sense of chaos and confusion [28] across the involved teams because everyone will not agree to the new plans. Development will progress smoothly if the involved teams and managers are willing to frequently communicate and share knowledge about the new changes in the product backlog. Our interview data showed that technical and experienced people are too busy to be approached, and some team members are so protective of their work due to fear of providing supporting. This confirms that there is **knowledge sharing problems (C3).**

Souza et al. [12] also noticed challenge of knowledge sharing when they used two scenarios to vindicate the issues of lack of awareness: (1) Manager's lack of awareness of evolving social dependencies, and (2) developers' lack of awareness of evolving code dependencies.

With this breakdown in communication and reduced knowledge circulations these conflicts (unplanned technical dependencies) will be unknown to some teams. It will be more difficult for these teams to resolve them since they do not know the sources. This will lead to spontaneous changes in the code, **reducing its quality (C4)**. This challenge was not revealed anywhere in the technical dependencies literature.

When **code quality challenge** is not resolved at this stage teams will most likely submit low quality code to the main branch for integration. This will lead to **integration problems (C5)**, where issues like incompatible components will arise, which may lead to a project re-planning. Majority of our interviewees expressed a strong dissatisfaction with the IBM Rhapsody tool when it comes to code merging. (**See section 5).** We think that the reason why other researchers in related work did not show anything to confirm merge challenge might be because they did not conduct a research in a setting where developers use Rhapsody to merge codes. Finally, there are findings of other researchers about technical dependencies that we did not confirm in this research. For instance Babinet and Ramanathan [1] saw system complexity as a challenge of technical dependencies, but our findings did not confirm it.

From the explanation above it is evident that the main challenges and categories of this study form a **technical dependency loop** that eventually lead to a **domino effect.**

Drawing our perspective on the discussion above on how technical dependency challenges create a domino effect we conclude that: (details on these conclusions can be found in the recommendation section)

- Identification and resolving of technical dependencies should be a continuous process rather than a planning phase activity
- Technical dependencies issues should be frequently communicated to all teams and managers
- Identifying technical dependencies should be a combined effort by all stakeholders on a given project
- Technical dependencies identified and not resolved at a given phase will spread to other phases. If they are resolved the technical dependencies loop will be broken for all further phases.
- Time should be set aside to identify, resolve and reflect on technical dependencies

Threats to Validity

A. **Internal validity**

In quantitative research, much emphasis seems to be placed on using random sampling to select interviewees to mitigate threats to internal validity [2]. On the contrary in a case study which is a strategy to qualitative research, Creswell [23] stated that there is no total agreement on the sample size of a qualitative research, but recommended that 3-5 interviewees be used for case study research. So, because we did not conduct quantitative research, using a convenience sample to select 9 interviewees for our research did not cause any threats to the internal validity [10] of our findings. However there might be the following threats:

- Not being able to come up with all of the important challenges of technical dependencies.
- Not being able to come up with all of the important challenges of communicating these technical dependencies

B. External validity

Contrary to threats to internal validity, since the strategy we are using in this qualitative research is a case study, which has the intention to enable analytical generalization, whereby the findings can be extended to other cases that have common characteristics [10]. Thus, the threat to external validity [2], [10] to the findings of this study is minimized.

In relation to section 6.1, it is also noteworthy that the fact that our findings are, to a large extent overlapping with the findings of other researchers, increases the external validity of the challenges we elaborated on in section 5.

C. Construct validity

The threat to construct validity [10] is also minimized in this study because there is an alignment in the interpretation of the ideas discussed in the interview questions between us and the interviewees at Ericsson AB. In addition, we conducted a pilot test on the interview questions between us, and two more people. We also ensured that the way in which we carried out our investigation was in accordance with our research questions.

D. Reliability

We mitigated threats to reliability [10] as follows:

- By having clear interview questions
- By coding the interviews data using thematic analysis [14]
- The academic supervisor assigned to us and our contact person at Ericsson reviewed the codes and themes we generated.

7 RECOMMENDATIONS

In this section, we present the recommendations based on the our interview finding and related work, that will help to mitigate the impact of the challenges of technical dependencies, and challenges of communicating technical dependencies in large-scale Agile software development. The recommendations are presented below following the order of the main challenges of this study.

7.1.1 Planning challenge, and team backlog priority challenge

1. Forming and Involving the Design Architects (DAs) team in the planning phase

At Ericsson this is not fully implemented across all teams. However our participants shared a view that the Design Architects team will be composed of software designers(coders who build components) from each XFT who will: (1) participate in the planning phase, (2)conduct regular meeting to share views on the issues that are happening in XFTs, and (3) share this information with XFTs.

We believe involving DAs in the planning phase will minimize planning challenges, because DAs know much about the code, so they will form a strong planning team that will be able to do the following:

- Identify technical dependencies
- Allocate tasks to teams with only necessary technical dependencies
- Reduce the knowledge sharing gap between the planning team and XFTs because DAs will directly communicate planning issues in the teams.
- Identify a team which has the expertise to accomplish a particular task, and whether to split a specific task and assign it to different teams, or assign the entire task to a single team.
- In cases where unplanned technical dependencies come up, DAs will be able to guide the XFTs to manage continuous changing of priorities in teams' backlog.

2. **Frequent checkpoint meetings**
To identify unplanned technical dependencies that come as a result of planning challenges.

We would also suggest they think about the use Ariadne. Ariadne is a plug-in for Eclipse that automatically show all the technical dependencies and the developers that have to coordinate with one another as a result of source-code dependencies [12].

7.1.2 Attitudes and knowledge sharing challenge (C3)

3. Broadening initiative
This initiative involves each XFT member to learn an additional role. For example, a software designer also develops competence in function testing. Our interviewees believe that broadening initiative will mitigate technical dependencies among members since they will have a variety of skills to address most of the dependencies issues that arise, on either an individual basis or on team basis.
We believe that broadening initiative will increase fast knowledge circulation because there will be no need for several people queuing to meet just limited people that have expertise in a specific role. However we caution organizations to provide regular short courses to avoid **broad competences** because this is one of the challenges we foresee that might result from broadening initiative in the long run.

4. Pool teams and competence broadening forums
We recommend after the implementation of the DAs teams, organization should extend this practice to all other roles in the XFTs. For example, formation of the test and integration pool comprising of tester from each XFT, documentation pool, and Scrum-master pool. These pools will help in developing best practices, coordinate processes, increase circulation of knowledge, and sharing of experiences in the XFTs. The pools should also be supplemented with competence broadening forums were XFTs meet pool members to share and solve issues.

6. Using Scrum-of-Scrums (SoS), whereby representatives of one XFT attend scrum meetings of other XFTs, thereby getting to know what they are working on and how they depended on one another. We believe SoS is good for sharing information between different XFTs, but there are other alternatives, such as, **Town Hall Meeting** (THM) **and Open Space Technology** (OST) [8], [29] that Agile practitioners believe work better than SoS. So, we would recommend the use of THM **and OST** for effective communication and coordination. OST creates an atmosphere where people can express their burning issues [29].

7. Managing silence by Sandberg and Mathiassen [5], to address the issue of people who are shy to contribute in meetings. One of the ways to get the shy people to contribute to meeting is by asking the person that is shy what his or her opinion is on the issue that is being discussed [5].

8. Early interaction across teams: Some interviewees suggested that managers should arrange interaction meetings before a project kicks off like, fika, parties or after work with an intention of getting teams to know each other and start interaction early enough. So when dependencies arise individuals are already aware of each other.

9. Change of teams physical structures
One participant suggested that teams should work in an open space instead of cubical rooms, and then create silent rooms for individual or teams that need total silence. This will increase physical interaction and knowledge sharing between individuals. Also teams doing similar work should be co-located to enable them easily access one another.

7.1.3 Code quality challenge (C4)

10. Automated script finding tools
This involves automatic sending of messages to individuals whenever changes are made in the same

area of code they are developing. This was suggested by our interviewees. They urge that this will make other developers to be aware of the changes and effects of these changes, so that they can start to adjust to these changes in their daily developments. We also believe this will help in finding sources conflicts, and also address test inefficiency issues.

We also recommend that managers, project managers, system designers, and DA should come to a consensus on the optimum number of system designers that can be in the same code.

7.1.4 Merge challenge (C5)

11. Continuous Integration (CI): Our interviewees mentioned that they have started frequent delivering of source-code to the main branch, and writing Trouble Reports (TR) to address **merge challenge.** We believe that frequent delivering of source-code, TR, and CI are good ways to address merge challenge.

We also suggest the replacement of IBM Rhapsody with another tool because of its limitation in the merging of source-code or teams should be given more training on how to use it

8 CONCLUSION AND FUTURE WORK

This paper reports the findings of a case study conducted at Ericsson AB, Sweden on the challenges associated with technical dependencies, and communicating technical dependencies, across large-scale Agile software development. The study was investigated using a qualitative research approach [2] which involved interviewing 9 participants at Ericsson AB. A thematic analysis of the interview data revealed: Planning, Teams backlog priority, Attitude and knowledge sharing, Code quality, and Merge challenge as the main challenges of this study. These challenges interact with each other forming a technical dependency loop, and lead to domino effect, during the development of a product. The magnitude of the domino effect will determine the quality of the final product. We suggested some recommendations such as broadening initiative, continuous integration, automated script finding among others, to mitigate the above challenges. We believe if the results of this study are put in practice there will be effective communication across teams, which will enable large scale companies realize the benefits of large scale agility.

Finally, we hope to see our study is replicated in another similar research setting, such as Volvo IT and others to see if their findings will be similar to what we came up with.

9 ACKNOWLEDGMENT

We would like to extend our thanks to Ericsson AB, and the Department of Computer Science and Engineering, Software Engineering Division, Chalmers | University of Gothenburg for facilitating this research. In particular we would like to thank the following people:

1. Anna Sandberg Börjesson: Change Program Driver at Ericsson AB for her support and guidance, and mostly being the lead organizer of interviews we conducted at Ericsson. In additional we also thank our interviewees for their contribution to this research,
2. Professor Michel R.V.Chaudron: Department of Computer Science and Engineering, Chalmers and Gothenburg University, Sweden, for the academic supervision and guidance throughout the development of this research.
3. Associate Professor Helena Holmström Olsson: Department of Computer Science, Malmö University, Sweden, for the guidance in the initial thesis proposition.

10 REFERNCES

[1] E. Babinet and R. Ramanathan, "Dependency Management in a Large Agile Environment," in *Agile, 2008. Agile '08. Conference,* 2008, pp. 401-406.

[2] J. W. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*: SAGE Publications, 2009.

[3] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Commun. ACM,* vol. 31, pp. 1268-1287, 1988.

[4] C. R. B. de Souza, D. Redmiles, G. Mark, J. Penix, and M. Sierhuis, "Management of interdependencies in collaborative software development," in *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on*, 2003, pp. 294-303.

[5] A. B. Sandberg and L. Mathiassen, "Managing Slowdown in Improvement Projects," *Software, IEEE,* vol. 25, pp. 84-89, 2008.

[6] B. J. E. Johansson and P.-A. Persson, "Reduced uncertainty through human communication in complex environments," *Cogn. Technol. Work,* vol. 11, pp. 205-214, 2009.

[7] P. Kettunen and M. Laanti, "Combining Agile software projects and large-scale organizational agility," *Softw. Process,* vol. 13, pp. 183-193, 2008.

[8] C. Larman and B. Vodde, *Scaling Lean and Agile Development*: Addison Wesley Professional, 2009.

[9] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM,* vol. 15, pp. 1053-1058, 1972.

[10] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Engg.,* vol. 14, pp. 131-164, 2009.

[11] J. Singer and N. G. Vinson, "Ethical Issues in Empirical Studies of Software Engineering," *IEEE Trans. Softw. Eng.,* vol. 28, pp. 1171-1180, 2002.

[12] C. R. d. Souza, S. Quirk, E. Trainer, and D. F. Redmiles, "Supporting collaborative software development through the visualization of socio-technical dependencies," presented at the Proceedings of the 2007 international ACM conference on Supporting group work, Sanibel Island, Florida, USA, 2007.

[13] M. Dainton and E. D. Zelley, *Applying communication theory for professional life: a practical introduction*: SAGE Publications, Incorporated, 2011.

[14] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology,* vol. 3, pp. 77-101, 2006/01/01 2006.

[15] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, *et al.*, "Manifesto for Agile software development," 2001.

[16] D. Leffingwell, *Scaling software agility: best practices for large enterprises*: Addison-Wesley Professional, 2007.

[17] K. Conboy and B. Fitzgerald, "Toward a conceptual framework of Agile methods: a study of agility in different disciplines," presented at the Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research, Newport Beach, CA, USA, 2004.

[18] P. Tomaszewski, P. Berander, and L. O. Damm, "From Traditional to Streamline Development—opportunities and challenges," *Software Process: Improvement and Practice,* vol. 13, pp. 195-212, 2008.

[19] K. Petersen and C. Wohlin, "The effect of moving from a plan-driven to an incremental software development approach with Agile practices," *Empirical Softw. Engg.,* vol. 15, pp. 654-693, 2010.

[20] E. Trainer, S. Quirk, C. d. Souza, and D. Redmiles, "Bridging the gap between technical and social dependencies with Ariadne," presented at the Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange, San Diego, California, 2005.

[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*: Addison-Wesley Longman Publishing Co., Inc., 1995.

[22] C. E. Shannon and W. Weaver, *The mathematical Theory of communication*: University of Illinois, 1949.

[23] J. W. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*: SAGE Publications, 2003.

[24] D. Koutsikouri, A. R. J. Dainty, and S. A. Austin, " Critical success factors for multidisciplinary engineering projects," presented at the Proceedings of the 22nd Annual ARCOM Conference, Birmingham, Uk, 2006.

[25] B. Boehm and R. Turner, "Management challenges to implementing Agile processes in traditional development organizations," *Software, IEEE,* vol. 22, pp. 30-39, 2005.

[26] C. Ciborra, *From control to drift: the dynamics of corporate information infrastructures*: Oxford University Press on Demand, 2000.

[27] G. Tjørnehøj and L. Mathiassen. (2008, Between control and drift: negotiating Improvement in a small software firm. *21(1),* 69-90.

[28] M. G. Weinberg, Ed., Quality software management. Dorset House Publishing: New York., 1997, p.^pp. Pages.

[29] C. Larman and B. Vodde, *Practices for Scaling Lean and Agile Development*: Addison Wesley Professional, 2010.

[30] M. E. Sosa, S. D. Eppinger, M. Pich, D. G. McKendrick, and S. K. Stout, "Factors that influence technical communication in distributed product development: an empirical study in the telecommunications industry," Engineering Management, IEEE Transactions on, vol. 49, pp. 45-58, 2002