Technical-, Social- and Process Debt in Large-Scale Agile: An Exploratory Case-Study

Antonio Martini[1(✉)], Viktoria Stray[1,2(✉)], and Nils Brede Moe[2(✉)]

[1] Department of Informatics, University of Oslo, Oslo, Norway
{antonima, stray}@ifi.uio.no
[2] SINTEF, Trondheim, Norway
nils.b.moe@sintef.no

Abstract. Large-scale agile projects bring inter-teams interaction challenges. Teams need to be autonomous, but often crosscutting concerns affect many teams. If the teams fail to collaborate on these concerns, the negative effects might hinder agility in the medium and long term. In other words, the organization and the system accumulate *debt*, on which the teams pay a high interest. Such debt must therefore be prioritized and "repaid" timely. We conducted a case study with interviews, observations and document analysis. Via both team- and large-scale retrospectives we investigated how teams coordinate and discuss Technical-, Social- and Process Debts.

Keywords: Large-scale software development · Coordination practices · Communication · Technical debt · Process debt · Social debt · Retrospective

1 Introduction

Large software companies strive to become more responsive in identifying and satisfying their customers' needs. One strategy for increasing responsiveness is the introduction of autonomous teams and agile software development [14]. However, when many agile teams are working towards the same goal in a Large-Scale Agile (LSA) project, a lot of coordination and management effort is required [13]. If each of the autonomous team in LSA setting works independently, team development processes and technical solutions would ultimately differ and may be highly disconnected from one another. Further, a high level of team autonomy and a need for constant delivering value to the customer, can lead to sub-optimal processes that might have short-term benefits, but generates a negative impact for the organization in the medium-long term. Examples of negative impact can be duplicated work, misunderstandings and integration problems.

In recent literature, a financial metaphor has been successfully used to describe such phenomena: aiming for short-term goals is equivalent to taking a *debt*, and the additional negative impact paid in the medium-long term is considered the *interest* that is paid on such debt. Although the metaphor has been used to describe prevalently technical issues (hence the term Technical Debt [4]), the debt metaphor has been used

R. Hoda (Ed.): XP 2019 Workshops, LNBIP 364, pp. 112–119, 2019.
https://doi.org/10.1007/978-3-030-30126-2_14

to describe other kinds of sub-optimal solutions, for example related to the social structure of the development community (Social Debt [16]) or to the development processes (Process Debt [1, 8]). However, how different types of Debt are identified and managed in LSA, is still an open question.

As an example, let us take what is called *Architectural Debt*: in LSA, system cross-cutting concerns (e.g. maintainability, usability and performance) and the consequent technical solutions, need to be envisioned at a higher level than from a single team perspective. As an example, if the concerns are not well separated in the system (which represents the Architectural Debt), several teams might find themselves working on the same shared component. Changing the same component in parallel, hinders the teams; they have to coordinate, merge conflicts and share the responsibility of the component. This, in the end, either creates a lot of overhead for the teams or causes the component quality to degrade, making future changes problematic (these effects are the interest paid on the debt). Communication is key to avoid these problems and might not happen if not supported by practices shared across the teams.

Recent literature in LSA has provided a better understanding on how teams coordinate; Dingsøyr et al. [5] describe 14 mechanisms for inter-team coordination in large-scale software projects. They found that coordination of work between teams influences teams' internal processes and how each team makes decisions. Nyrud and Stray [11] identified 11 coordination mechanisms in a large-scale agile project and found retrospective meetings to be important for continuous improvement of the inter-team coordination mechanisms. Further, managers need to be sensitive to the coordination needs as they change over time in large programs [10].

However, to the best of our knowledge, there are no related studies addressing the challenge of balancing the management of technical-, social- and Process Debt in LSA. We aimed to understand which types of debt that require inter-team coordination in LSA by answering the following research question:

RQ: What types of debt are elicited and discussed on team level and inter-team level in a large-scale agile project?

2 Technical-, Social- and Process Debt

Different types of debts have been researched. However, Technical Debt (TD) is the one for which the most literature is available. The definition of TD is [2]:

> *" […]a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. […]".*

Different types of TD have been identified. The most common ones are [8]:

- **Code Debt,** which is related to sub-optimal code solutions. For example, not declaring a variable that later requires manually changing all the instances in the code.
- **Architectural Debt** is regarded as sub-optimal solutions with respect to an ideal architecture. For example, a monolithic architecture with many dependencies across

modules requires teams to change the code in many places for any change. This might require more time to deliver features and might introduce bugs.
- **Test Debt** is regarded as lack or sub-optimal tests. For example, low code coverage or the lack of structured and automated tests can be considered test debts.
- **Documentation Debt** is the lack or sub-optimal documentation. An example could be the lack of description on how APIs work, which hinders developers when correctly accessing modules, components or services across the system.
- **Infrastructure Debt** is related to sub-optimal solutions in the development environment. For example, buggy knowledge management tools.

Other types of debt (non-technical) have also been identified in literature as causing negative impact, namely Social Debt and Process Debt.

As for **Social Debt**, it is referred to as *"the presence of sub-optimality in the development community, which causes a negative effect"* [16]. An example of Social Debt is the lack of proper communication among key parts of the organizations (e.g. between development and operations). Another example is having an architecture team that is disconnected from the team and therefore might suggest architectural solutions that are not realistic, as they do not take into consideration details and requirements elicited during implementation.

On the other hand, **Process Debt** is mentioned as a type of debt that needs to be managed [1, 8], but there exists no current definition for it. We therefore use our own operational definition based on the ones reported for the other types of debt. We define Process Debt as *"a sub-optimal activity or process that might have short-term benefits, but generates a negative impact in the medium-long term"*. An example of Process Debt might be that teams conduct stand-up meetings where the focus is reporting status so that the leader knows what is going on [15]. The short-term benefit is that the team members satisfy their leader's need for knowing project status. However, a long-term negative impact is that the meeting centers around reporting and not about sharing knowledge and solving dependencies which is more valuable in the medium-long term. Another example is a large-scale project having the presence of many meetings to coordinate, which might seem important for solving dependencies, but disrupts the development work compromising the project efficiency [3, 15].

3 Research Methodology

Since the goal of this research was to explore and provide insight into the phenomenon of both technical and non-Technical Debt in LSA, we designed a case study [18] to observe the various types of debt in practice. We chose a large-scale project that develops a new platform supporting public transportation as our case. The project has thirteen development teams ranging between five and fourteen team members working towards the same products.

Understanding the design or implementation constructs that are expedient in the short term, but that can make future changes more costly or impossible is a complex problem. There is a need to understand challenges and improvement suggestions together with what is working well. Further, there is a need to get many stakeholders of

the LSA project together as no one in an LSA project has the full overview of the situation. Also, the teams need to discuss issues that (potentially) negatively impact one or more teams and involve more than one team (inter-teams) such as shared components.

Conducting retrospective meetings is an important and popular practice in agile software development [17], and applied both on team-level and in large-scale agile [6]. The meetings are utilized for improving the way of working, and participants often discuss past challenges and how to overcome them to work better together in the future [7]. Therefore, we chose retrospectives as the primary source of our data collection for studying our research question. We chose to study team-level retrospectives (see Fig. 1) and a large-scale retrospective. Additionally, we conducted two informal group interviews to prepare for the retrospective sessions. All the six reports from the meetings were imported into NVivo. We analyzed the meetings by categorizing the reported issues into the different type of debts described in the background section.

Fig. 1. Issues discussed during one of the team retrospectives using DAKI

- ***Team Retrospectives***: In November 2017 we facilitated a retrospective meeting in Team Alpha (see Fig. 1). This retrospective meeting had eight team members present. We also collected four reports from their previous retrospective meetings. These reports included what had been reported as positive and negative issues in addition to action items (often with a person responsible for following up the item). These retrospectives had been held between December 2016 and September 2017.
- ***Large-Scale Retrospective***: We facilitated a retrospective meeting with project leaders, product owners and the team leaders in the LSA project; 13 people in total. The focus in this retrospective was on a large delivery they had been working on from May to November 2nd, 2017. To elicit the relevant problems we chose to use an exercise where participants discuss issues that need to be dropped, added, kept or improved (DAKI) [12]. This exercise uses four quadrants where participants can

place issues and is good to use when there is a high number of participants and issues. We then facilitated a discussion of the most urgent items by using the technique Lean Coffee [9] (Table 1).

Table 1. Data collection

Data	Explanation	Number
Informal group interviews	Questions about the teams, how they were working, the roles and their retrospective meetings	2
Retrospective reports Retrospective meeting in one team Large-scale retrospective	We collected reports from Team Alpha We facilitated a retrospective in Team Alpha We facilitated a retrospective meeting with representatives from teams in the LSA project	4 1 1

4 Results

We observed which different types of debt were discussed in team retrospectives compared to a large-scale retrospective. This would tell us if some types of debts seem to be more team issues (and would not require coordination) rather than inter-team issues.

We therefore report such comparison by counting the number of issues for each type of debt in Table 2. First, we outline the difference between Process, Social Debt and overall Technical Debt. Then, we show the number of issues related to the sub-types of Technical Debt. We omit Code Debt, as we did not find any Code Debt issue discussed.

Table 2. Number of issues discussed in team and large-scale retrospectives

Debt type	Team retrospectives	Large-scale retrospectives
Process debt	24	17
Social debt	37	15
Technical debt	26	11
Architecture debt	6	2
Documentation debt	7	2
Infrastructure debt	7	2
Test debt	6	5

We report, in Table 3, the issues that were discussed in both team retrospectives and LS retrospectives, as well as some of the issues that were instead discussed only on a team level. This would show the difference between which issues were autonomously addressed and which ones required coordination. Below the table, we list the only two issues that were brought up during the large-scale session only (related to Test Debt).

Table 3. Issues that were discussed by teams only and issues that were re-proposed for inter-team discussion and coordination

Debt type	Re-proposed at inter-team level	Team retrospectives only
Architecture debt	• Unstable APIs (teams modifying often) • Unclear module responsibilities (more structure needed)	• Deployment issues
Documentation debt	• Need for spread the documentation across teams	• Documentation issues for specific modules
Infrastructure debt	• Problems related to Jira and its usage for stories and epics	• Issues related to a specific knowledge management tool used by the team
Test debt*	• More automated tests • More test follow-up	• Test coverage • Specific test structure
Process debt	• Have more demo sessions • More and better structured planning • More meetings preparation and effectiveness (too many people) • Specific release event not well executed • More pair-programming • Agile definition (e.g. sprint definition)	• Issues related to a specific team process (e.g. team planning)
Social debt	• Team autonomy • Involvement and synchronization with leadership • Sync with POs, PMs, etc • Information and knowledge across teams • Common goals • Communication and involvement of UX developers	• Slack usage and content • Team specific roles (e.g. specific module testing responsibility) • Team specific competences (e.g. UX, design)

*Test Debt issues that were discussed on Large-Scale Retrospectives only:
- Better definition for acceptance criteria for tests
- Better end-to-end tests

5 Discussion and Limitations

The aim of this work was to understand which types of debt that require inter-team coordination in LSA by answering the following research question: What types of debt are elicited and discussed on team level and inter-team level in a large-scale agile project?

5.1 What Types of Debt Are Elicited and Discussed on Team Level and Inter-Team Level in a Large-Scale Agile Project?

This study provides an initial source of evidence on the concepts of Technical-, Social and Process Debt, and that inter-team coordination is required to tackle the various types of debt. First, we found that many of the Process Debt issues were re-proposed in the inter-team discussions, which implies that Process Debt needs coordination to be solved, potentially even more than other types of debt. Second, teams discuss heavily Social Debt issues in team retrospectives, and part of the issues are discussed again in large agile retrospectives, especially related to team autonomy, leadership and communication across teams. Third, Technical Debt is discussed in team retrospectives, but only part of it is re-proposed for inter-team discussion.

We did not find Code Debt discussed in retrospectives at all. As for Architecture, Documentation, and Infrastructure, inter-team coordination seems to be necessary when it comes to APIs, documentation used across the teams and the usage of the tools that are used by more teams.

We found a special case related to Test Debt, as some issues were re-proposed (automation, structure), while others were discussed only on a team level. In addition, some of the Test Debt-related issues (e.g. related to acceptance criteria, end-to-end tests) were discussed only on an inter-team level: it seems that such issues were elicited thanks to the joint discussion among teams.

5.2 Limitations and Future Work

In this paper, we used retrospectives to answer our research question. However, other sources of data, for example from other forms of communication, should be analyzed to complement the findings. For example, it could be that architectural issues are discussed in meetings that are more technical rather than in retrospectives. Furthermore, more cases should be analyzed to understand if the results are similar in other large-scale agile projects or if the studied project was a special case.

Acknowledgements. This work was partially supported by the Research Council of Norway through grant 267704 and by the companies Kantega, Knowit, Storebrand and Sbanken.

References

1. Alves, N.S.R., et al.: Towards an ontology of terms on technical debt. In: 2014 Sixth International Workshop on Managing Technical Debt, pp. 1–7 (2014). https://doi.org/10.1109/MTD.2014.9
2. Avgeriou, P., et al.: Managing technical debt in software engineering (dagstuhl seminar 16162). In: Dagstuhl Reports. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2016)
3. Bass, J.M.: Scrum master activities: process tailoring in large enterprise projects. In: 2014 IEEE 9th International Conference on Global Software Engineering, pp. 6–15. IEEE (2014)
4. Cunningham, W.: The WyCash portfolio management system. In: ACM SIGPLAN OOPS Messenger, pp. 29–30. ACM (1992)

5. Dingsøyr, T., et al.: Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. Empirical Softw. Eng. **23**(1), 490–520 (2018)
6. Dingsøyr, T., Mikalsen, M., Solem, A., Vestues, K.: Learning in the large - an exploratory study of retrospectives in large-scale agile development. In: Garbajosa, J., Wang, X., Aguiar, A. (eds.) XP 2018. LNBIP, vol. 314, pp. 191–198. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91602-6_13
7. Kniberg, H.: Scrum and XP from the Trenches. Lulu. com (2015)
8. Li, Z., et al.: A systematic mapping study on technical debt and its management. J. Syst. Soft. **101**, 193–220 (2015). https://doi.org/10.1016/j.jss.2014.12.027
9. Österberg, M., Esni, B., Rabiee, S., Majkowska, Z.: Spotify Retro kit (2017)
10. Moe, N.B., et al.: To schedule or not to schedule? an investigation of meetings as an inter-team coordination mechanism in large-scale agile software development. IJISPM **6**(3), 45–59 (2018)
11. Nyrud, H., Stray, V.: Inter-team coordination mechanisms in large-scale agile. In: Proceedings of the XP2017 Scientific Workshops, pp. 1–6. ACM Press (2017). https://doi.org/10.1145/3120459.3120476
12. Caroli, P., Caetano, T.: Fun retrospectives: activities and ideas for making agile retrospectives more engaging. http://www.caroli.org/product/fun-retrospectives-activities-and-ideas-for-making-agile-retrospectives-more-engaging/
13. Petersen, K., Wohlin, C.: The effect of moving from a plan-driven to an incremental software development approach with agile practices. Empirical Softw. Eng. **15**(6), 654–693 (2010)
14. Stray, V., et al.: Autonomous agile teams: challenges and future directions for research. In: 19th International Conference on Agile Software Development: Companion, XP 2018. ACM, New York (2018). https://doi.org/10.1145/3234152.3234182
15. Stray, V., et al.: Daily stand-up meetings: start breaking the rules. IEEE Software. (2018). https://doi.org/10.1109/MS.2018.2875988
16. Tamburri, D.A., et al.: What is social debt in software engineering? In: 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), pp. 93–96. IEEE (2013)
17. Version One: 12th State of Agile Report. https://www.infoq.com/news/2018/04/state-of-agile-published
18. Yin, R.K.: Case Study Research: Design and Methods. Sage, Thousand Oaks (2009)