Technical Debt in Large-Scale Distributed Projects: An Industrial Case Study

Armando Sousa and Lincoln Rocha
Federal University of Ceará
Fortaleza, Brazil
armando.sousa@alu.ufc.br, lincoln@dc.ufc.br

Ricardo Britto
Ericsson AB
Blekinge Institute of Technology
Karlskrona, Sweden
ricardo.britto@{ericsson.com, bth.se}

Zhixiong Gong and Feng Lyu
Blekinge Institute of Technology
Karlskrona, Sweden
{zhgo15, felv15}@student.bth.se

***Abstract*—Technical debt (TD) is a metaphor that reflects the technical compromises that sacrifice the long-term health of a software product to achieve short term benefit. It is important to manage TD to avoid software degradation. In large-scale distributed projects, technical debt management (TDM) becomes more complex and challenging. There is a lack of empirical studies on the TD accumulation in large-scale distributed projects. Then, to address this gap, we conducted a case study in Ericsson (a European Telecom Company) to identify the relationship between TD accumulation and factors such as task complexity, lead time, total of developers, and task scaling. We used data from 33 projects extracted from managerial documents to conduct a regression analysis. We also conducted interviews with seniors developers of the team to interpret the results. We found out that Task Complexity has a strong relationship (p-value = 5.69×10^{-5}) with Technical Debt, while Global Distance was mentioned by the interviewees as a relevant factor for TD accumulation (although not statistically significant in our regression analysis). Practitioners should consider avoiding complex/big tasks, breaking down big tasks into small ones (if possible). We also plan to analyze other projects in this company to confirm our findings further.**

***Index Terms*—technical debt management, large-scale, global software engineering**

I. Introduction

Organizations around the world develop software in a globally distributed way (Global Software Engineering – GSE) to achieve benefits such as reduced time-to-market and access to skilled people all over the world [1]–[3]. However, geographical, temporal, and cultural distances amplify the difficulties associated with coordination and communication in GSE projects [1].

It is often the case that GSE projects involve a large number of people (large-scale projects[1]). The combination of scale and global distribution may lead to problems, such as more software defects [5], schedule and budget overruns [6], and make it challenging to manage Technical Debt (TD) [7].

TD reflects technical compromises to achieve short-term benefit at the cost of hurting a software product's long-term health, which puts future development and maintenance at high potential risk [8]. TD refers to any incomplete, immature, or inadequate artifact in the software development life cycle affecting subsequent development and maintenance activities, which is treated as a type of debt that the developers owe the system [9].

To keep TD accumulation under control, Technical Debt Management (TDM) is required throughout the development process. Part of TDM includes activities preventing potential TD from being incurred. Meanwhile, TDM also includes activities dealing with the accumulated TD to make it visible and controllable and balance the software project's cost and value. TDM in large-scale GSE projects can be more complex. For example, it may be more challenging to ensure a common understanding of TD and TDM across multiple sites [7]. Moreover, factors such as distance [10], [11] are known to be associated with TD accumulation. Also, TD-related decisions are often not systematically used. There are no generic approaches used in the industry that facilitate systematic TDM (e.g., TD decisions are often not even explicitly captured) [12].

To the best of our knowledge, there is no investigation on the TD accumulation in large-scale globally distributed software projects. Given the relevance of the topic for both research and industry, we have made an attempt to fill the existing gap through conducting an industrial case study in Ericsson, a company that develop hardware and software telecommunication solutions.

In this paper, we report the findings of our investigation, which address the following research question: *What factors are related to the TD accumulation in large-scale GSE projects*?

The remainder of this paper is organized as follows: Section II describes the background and related work. Section III presents the research design. Section IV presents the results and discussions. Validity threats are discussed in Section V. Finally, we provide our conclusions and view on future work in Section VI.

II. Background and Related Work

In the GSE-related literature, the following topics stand out: the global distance (GD) between the teams, the form of communication between the project participants, and the developers' level of maturity.

[1]Dikert et al. [4] define as large-scale software projects that involve at least 50 human resources – not necessarily only developers, but also other staff collaborating in software development – or at least six teams.

GD measures the overhead of cooperation and coordination in communication between several sites. Effective communication between distributed sites is crucial for a successful distributed project [13]. However, distance negatively affects communication, which in turn reduces coordination effectiveness [14]. Thus, increasing the risk of incurring TD. Kazman et al. [15] used a model approach to analyze the software architecture as a set of design rules spaces. Heikkilä et al. [16] explored how hard communication is for the practitioners in large-scale globally distributed software projects.

Team Maturity (TM) implies an increased capability of controlling and managing TD, with important historical data available for development teams to quantitatively manage and control key projects as well as organizational processes [17]. [18] shows that high maturity can reduce cycle time and development effort suggests a lower TD.

Another aspect that makes software development more difficult is task complexity (TC). Alzaghoul et al. [19] found that higher complexity may indicate higher rework costs. As a result, increased complexity might lead to an increase in TD.

When a developer identifies a debt, documenting the debt helps to manage TD systematically. Formal documentation can make the TD traceable and increase the effectiveness of TDM [20]. Guo et al. [21] have proposed an approach to TDM based on systematic monitoring for each incremental release of a software product.

TD monitoring and TD repayment are two of the most important TDM activities, which helps managers to see the changes in the cost and benefit of unresolved TD over time [22]. Seaman and Guo [9] suggest that through monitoring, development teams could find a proper guide using a TD list as the center of monitoring the status of TD. TD repayment has a strong relationship with TD measurement and monitoring because to repay the debt, the team should check how urgent the debt is and decide when to repay the debt, as mentioned. The perception of TD through monitoring and repayment was studied by Besker et al. [23] that explored the perception of TD in the software development cycle. They did a survey to estimate the time lost caused by the Technical Debt accumulation during the software life cycle.

Digkas et al. [24] conducted a case study on 57 open-source Java projects from the Apache ecosystem to investigate how developers fix issues and payback TD over time. They found that a small portion of the issue types is responsible for the largest amount of TD repayment.

To gain a better understanding of TD in industrial setups, Rios et al. [25] conducted a tertiary study to look into the state of practice in several companies to understand the cost of managing TD, how maturity is managed in TD, what tools are used to track TD, and how a TD tracking process is deployed in practice.

Despite covering several TD issues, the reviewed papers do not focus on studying the TD accumulation in large-scale GSE projects. This paper fills the gaps mentioned above by employing an exploratory case study in a real and large-scale GSE project.

III. Research Design

To address our research question, we have conducted an exploratory longitudinal case study [26].

A. The Case and Unit of Analysis

The **case and unit of analysis** is a telecommunication software product developed by Ericsson. This software has been evolving for more than 24 years with several technological changes, such as the inclusion of additional programming languages (Java in addition to C++) and a change in development methodology from plan-driven to agile practices.

The product is developed in a geographically distributed fashion and includes (or has included) sites located in the USA, Sweden, Italy, and India. It involves cross-functional teams that have from 4 to 7 developers and use agile practices. Project managers use a mix of agile and plan-driven practices to manage and coordinate teams across sites. The teams are responsible for tasks such as product customization (PC), bug fixing, and product refactoring. PCs are carried as independent projects that may take from 1 to 6 months.

The data collected and used in our investigation comprises the period from January 2013 to August 2016. It includes only PC tasks because they have the most significant impact and value for the company's customers who use the product.

B. Variables

In this paper, the following **variables** were used for analysis: Technical Debt (TD), Task Complexity (TC), Lead Time (LT), Global Distance (GD), Total Developers (DV), Task Scaling (TS), and Team Maturity (TM). We selected these variables due to their relevance in GSE contexts and also due to the possibility to measure them in the investigated case [27], [28]. Table I presents a description of the investigated variables.

C. Data Collection and Data Analysis

To collect the data associated with the investigated variables, we employed three **data collection** methods: archival research, semi-structured interviews, and repository mining.

We employed **archival research** to measure LT, TS, from the 33 investigated PCs. TD was measured through **repository mining**, while TC and TM were measured through interviews in a previous investigation conducted by the second author of this paper [28].

To analyze this data, we employed the **data analysis** method hierarchical multiple regression analysis, aiming at understanding the relationship between the selected factors and TD accumulation. More details can be viewed in the replication kit available in the data repository[2] of this study.

To support the interpretation of the regression analysis results, we conducted two **semi-structured interviews**: we first interviewed a Software Architect (SA1) in June 2017.

[2]https://github.com/Technical-Debt-Large-Scale/tdmls

TABLE I
STUDY VARIABLES

ID	Name	Type	Data Collection	Description
TD	Technical Debt	Dependent	Repository Mining	Is the amount of dollars calculated by using SonarQube needed to fix all problems (duplication, violations, comments, coverage, complexity, bugs, bad design) in the code base.
TC	Task Complexity	Independent	Interviews	Is the parameter used to describe how complex the task. Each PC was estimated by a positive integer (complexity points) [28].
LT	Lead Time	Independent	Archival Research	Is the total time needed to deliver a task, counted by days.
TS	Task Scaling	Independent	Archival Research	Is the capacity of a task resizes according to the increase in demand for this task.
GD	Global Distance	Independent	Archival Research	Is the metric that measures the complexity of communication between sites, which represents the overhead of cooperation and coordination when more than one site is involved [29].
DV	Total Developers	Independent	Archival Research	Is the number of developers involved in the development of each task.
TM	Team Maturity	Independent	Interviews	Is the parameter to describe the level of how a team can deliver the product independently [28].

TABLE II
FACTORS CORRELATED TO TD

Factors	Impact	Spearman's ρ	p-value	Correlated
LT	Positive	0.486	5.00×10^{-3}	YES
TC	Positive	0.650	5.69×10^{-5}	YES
DV	Positive	0.505	3.00×10^{-3}	YES
TS	Negative	-0.439	1.20×10^{-2}	YES
TM	N/A	-0.135	4.62×10^{-1}	NO
GD	N/A	0.034	8.55×10^{-1}	NO

In a second moment, we conducted a group semi-structured interview with two other software architects and another a semi-structured interview with 2 Software Architects (SA2 and SA3) in January 2018. Each meeting took approximately one hour. All interviewees had more than ten years of large-scale GSE experience. The questions can be viewed in the data repository of this study.

The semi-structured interview results were analyzed using content analysis since it is a systematic and rule-guided technique used for analyzing all sorts of textual data. It provides a brief and broad description of the phenomenon and allows researchers to enhance the understanding of raw data [30].

IV. RESULTS AND DISCUSSION

This section presents and discuss the results of the conducted regression analysis. We first present the results of checking the assumptions of the employed method, which is followed by the actual results of the analysis and discussion.

A. Regression Analysis Assumptions

We created a box-plot to analyze the TD values among all involved sites. As a result we identified two outliers. Only one was removed since the other was deemed as relevant for the analysis.

First, to check correlation among the selected features and TD (and identify linear relationships), we used Spearman's rank coefficient (Table II). As a result, we identified that four features (LT, TC, DV and TS) correlated with TD (p-value < than 0.05).

TABLE III
TESTING MULTICOLLINEARITY - (VIF, TOLERANCE)

Model	LT	TC	DV	TS
model1	(1,1)	-	-	-
model2	(1.12, 0.89)	(1.12, 0.89)	-	-
model3	(1.60, 0.62)	(1.14, 0.88)	(1.57, 0.64)	-
model4	(1.76, 0.57)	(1.50, 0.67)	(1.82, 0.55)	(1.51, 0.66)

Second, to further investigate the nature of the relationship between TD and the factors with significant correlation, we used partial regression plots [31]. As a result, the plots confirmed that there is some level of linearity between TD and LT, TC, DV and TS.

Third, we tested for auto-correlation using the Durbin-Watson test. If the Durbin-Watson test's value is between 1.5 and 2.5, there is no linear auto-correlation in the data. The Durbin-Watson values in our tests are the following: lead time = 1.614, task complexity = 2.155, total developers=1.230, task scaling = 1.727, and technical debt=2.041, which were acceptable. So, the residuals are independent in our data.

Fourth, we tested the normality of the residuals. To do so, we used P-P plots. The points on the plot remain close to the diagonal line, which means residuals are normally distributed.

Fifth, we tested the assumption of homoscedasticity. To do so, we used the Breusch-Pagan test. The Lagrange multiplier statistic was 2.326 and p-value was 0.676, i.e., the assumption of homoscedasticity was met.

Finally, to verify the presence or absence of multicollinearity, we used Tolerance/VIF (Variance Inflation Factor). The tolerance of independent variables should be greater than 0.1 and VIF less than 10. Table III shows that the tolerance values in our study are all greater than 0.1 and the VIF values all less than 10.

As a result, the following regression model was presented where TD is the dependent variable and LT, TC , DV, and TS are the predictors. Equation (1) presents the resulting model used in our analysis:

$$TD = 1048.31 + 311.52 * LT + 3234.82 * TC + 1241.58 * DV - 1495.39 * TS \quad (1)$$

B. *Results and Discussions of Factors related to TD*

According to the interviewees, all four factors (TC, LT, DV, and TS), relate to TD accumulation. Although the interviewees mentioned that TM and GD are also somehow related to TD, we could not confirm this in the conducted regression analysis.

Architect SA1 said that task complexity (TC) has a strong relation to TD since complicated tasks tend to have more debt. However, it was hard for the architect to judge exactly what can be seen as a complicated task. The SA2 confirmed this:

SA2: "*what is a complex task is hard to say, when a task contains a lot of lines of code, but from the functional perspective, it is very easy to build and will not create any debts at all, do we still think it has low complexity?*"

Alzaghoul and Bahsoon [19] found that increase in a software's complexity leads to an increase in TD. If the complexity increases due to changes in a software's structure, the dependencies between different parts of the software may become more complex as well, which may cause potential extra work to maintain the software.

We also learned that the longer it takes to complete a product customization developemnt cycle, the higher the TD. Besker et al. [32] identified that the shorter lead times (LT) can help to maintain costs under control, through using good planning between the moment of the product customer's order until the delivery can offer many advantages like cost reduction.

Regarding task scaling (TS), we observed that as the size of TS increases, the amount of TD tends to decrease. This looks counter-intuitive at first. For example, Guo et al. [21] identified that for large systems developed in a collocated manner, it is easy to lose track of delayed tasks or to misunderstand their impact. In our case, which does not go in the same direction of Guo et al., we believe that the observed relationship may relate to the fact that tasks with high TS often involved senior developers to support newcomers, which might have lead to lower TD in those cases.

In the case of total developers (DV), the total number of developers in a software project is critical factor in GSE projects, due to the difficulty communicate when there is a large amount of people [10], [1].

Regarding global distance (GD), although we did not identify a statistical significant relationship between GD and TD accumulation, software architect SA2 mentioned:

SA2: "*The worst case is that people working with the same functionality are sitting in different places and doing different phases of the work.*"

Architect SA2 also mentioned team maturity (TM), although it was not statistically significantly related to TD in our results. Although not significant, we identified that maturity tends to relate to TD accumulation (the higher the maturity, the higher the TD). After investigating our dataset, we identified that the largest and most complex tasks tend to be attributed to the mature teams in the investigated case. This means that the observed correlation between TM and TD is likely affected by the complexity and the size of the PCs.

V. Validity Threats

The validity threats associated with our investigation are discussed using the categories internal and external validity described by Runeson and Höst [26].

In relation to **internal validity**, one limitation is that we were able to investigate a subset of factors that potentially relate to TD accumulation. Other factors can still be studied, such as social, cultural, and other technical factors not evaluated in this study.

Regarding **external validity**, since we employed the case study method, our findings are strongly bound by our research context. In addition, the investigated case involved only one product in one company. To mitigate this threat, we described the context of our study in as much detail as possible so that the readers can identify if the context of our investigation is similar to theirs and reuse our findings whenever applicable.

VI. Conclusions and Future Work

This paper reports the results of a case study conducted in Ericsson that aimed at investigating the accumulation of TD in a large-scale globally distributed software project.

The overall conclusion is that TD accumulation strongly correlates with specifics factors of GSE projects. We believe that the process of TDM becomes more complex in globally distributed projects with different sites and different teams. Thus, a suitable TDM process must consider the GSE factors that correlate with TD accumulation, which we plan to investigate in a future study.

Our investigation has some implications for both researchers and practitioners. Regarding researchers, we believe that it is still necessary to conduct similar research in other companies to learn more about the accumulation of TD in large-scale globally distributed software projects.

We identified that task complexity is the factor most related to TD accumulation. Thus, practitioners should be aware of this and try to avoid complex projects and subdivide them into less complex projects as much as possible to prevent TD accumulation.

Finally, we plan to continue investigating other cases in this company to strengthen the empirical evidence reported herein.

References

[1] J. D. Herbsleb and D. Moitra, "Global software development," *IEEE software*, vol. 18, no. 2, pp. 16–20, 2001.

[2] E. Conchúir, P. J. Ågerfalk, H. Holmstrom, and B. Fitzgerald, "Global software development: Where are the benefits?" *Communications of the ACM*, vol. 52, no. 8, pp. 127–131, aug 2009.

[3] N. Ramasubbu, M. Cataldo, R. K. Balan, and J. D. Herbsleb, "Configuring global software teams: A multi-company analysis of project productivity, quality, and profits," in *Proceedings of the 33rd International Conference on Software Engineering - ICSE'11*, 2011, pp. 261–270.

[4] K. Dikert, M. Paasivaara, and C. Lassenius, "Challenges and success factors for large-scale agile transformations: A systematic literature review," *Journal of Systems and Software*, vol. 119, pp. 87–108, 2016.

[5] J. A. Espinosa, N. Nan, and E. Carmel, “Do gradations of time zone separation make a difference in performance? a first laboratory study,” in *Second IEEE International Conference on Global Software Engineering - ICGSE'07.*, Aug 2007, pp. 12–22.

[6] J. D. Herbsleb and A. Mockus, “An empirical study of speed and communication in globally distributed software development,” *IEEE Transactions on Software Engineering*, vol. 29, no. 6, pp. 481–494, June 2003.

[7] R. Bavani, “Distributed agile, agile testing, and technical debt,” *IEEE Software*, vol. 29, no. 6, pp. 28–33, Nov 2012.

[8] W. Cunningham, “The wycash portfolio management system,” in *Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum)*, ser. OOPSLA '92. New York, NY, USA: ACM, 1992, pp. 29–30.

[9] C. Seaman and Y. Guo, “Measuring and monitoring technical debt,” in *Advances in Computers*. Elsevier, 2011, vol. 82, pp. 25–46.

[10] E. Carmel and R. Agarwal, “Tactical approaches for alleviating distance in global software development,” *IEEE Software*, vol. 18, no. 2, pp. 22–29, 2001.

[11] D. A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, “What is social debt in software engineering?” in *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 2013, pp. 93–96.

[12] J. Holvitie, V. Leppanen, and S. Hyrynsalmi, “Technical debt and the effect of agile software development practices on it-an industry practitioner survey,” in *2014 Sixth International Workshop on Managing Technical Debt*. IEEE, 2014, pp. 35–42.

[13] Y. Yao, S. Huang, L. Jie, and X. ming Liu, “Structural characteristic of large-scale software development network,” in *2010 2nd International Conference on Computer Engineering and Technology*. IEEE, 2010.

[14] V. Casey and I. Richardson, “Uncovering the reality within virtual software teams,” in *Proceedings of the 2006 International Workshop on Global Software Development for the Practitioner*, ser. GSD '06. New York, NY, USA: ACM, 2006, pp. 66–72.

[15] R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyev, V. Fedak, and A. Shapochka, “A case study in locating the architectural roots of technical debt,” in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, may 2015.

[16] V. Heikkila, M. Paasivaara, C. Lasssenius, D. Damian, and C. Engblom, “Managing the requirements flow from strategy to release in large-scale agile development: a case study at ericsson,” *Empirical Software Engineering*, vol. 22, no. 6, pp. 2892–2936, 2017, cited By 1.

[17] D. Falessi, M. A. Shaw, F. Shull, K. Mullen, and M. Stein, “Practical considerations, challenges, and requirements of tool-support for managing technical debt,” in *Proceedings of the 4th International Workshop on Managing Technical Debt*, ser. MTD '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 16–19.

[18] D. E. Harter, M. S. Krishnan, and S. A. Slaughter, “Effects of process maturity on quality, cycle time, and effort in software product development,” *Manage. Sci.*, vol. 46, no. 4, pp. 451–466, apr 2000.

[19] E. Alzaghoul and R. Bahsoon, “Evaluating technical debt in cloud-based architectures using real options,” in *2014 23rd Australian Software Engineering Conference*, April 2014, pp. 1–10.

[20] S. Das, W. G. Lutters, and C. B. Seaman, “Understanding documentation value in software maintenance,” in *Proceedings of the 2007 Symposium on Computer Human Interaction for the Management of Information Technology*, ser. CHIMIT '07. New York, NY, USA: ACM, 2007.

[21] Y. Guo, R. Spinola, and C. Seaman, “Exploring the costs of technical debt management – a case study,” *Empirical Software Engineering*, vol. 21, no. 1, pp. 159–182, 2016, cited By 7.

[22] Z. Li, P. Avgeriou, and P. Liang, “A systematic mapping study on technical debt and its management,” *J. Syst. Softw.*, vol. 101, no. C, pp. 193–220, mar 2015.

[23] T. Besker, A. Martini, and J. Bosch, “The pricey bill of technical debt: When and by whom will it be paid?” in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sept 2017, pp. 13–23.

[24] G. Digkas, M. Lungu, P. Avgeriou, A. Chatzigeorgiou, and A. Ampatzoglou, “How do developers fix issues and pay back technical debt in the apache ecosystem?” in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, mar 2018.

[25] N. Rios, M. G. de Mendonça Neto, and R. O. Spínola, “A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners,” *Information and Software Technology*, vol. 102, pp. 117–145, oct 2018.

[26] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley Sons, 2012.

[27] R. Britto, D. Šmite, and L.-O. Damm, “Experiences from measuring learning and performance in large-scale distributed software development,” in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2016, p. 17.

[28] R. Britto, D. Smite, and L.-O. Damm, “Software architects in large-scale distributed projects: An ericsson case study,” *IEEE Software*, vol. 33, no. 6, pp. 48–55, 2016.

[29] A. Avritzer, S. Beecham, R. Britto, J. Kroll, D. S. Menasche, J. Noll, and M. Paasivaara, “Extending survivability models for global software development with media synchronicity theory,” in *Global Software Engineering (ICGSE), 2015 IEEE 10th International Conference on*. IEEE, 2015, pp. 23–32.

[30] P. Mayring, “Qualitative content analysis: theoretical foundation, basic procedures and software solution,” 2014.

[31] J. Fox, *Applied regression analysis and generalized linear models*. Sage Publications, 2015.

[32] T. Besker, A. Martini, and J. Bosch, “Technical debt triage in backlog management,” in *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE, 2019, pp. 13–22.