The Sources and Approaches to Management of Technical Debt: A Case Study of Two Product Lines in a Middle-Size Finnish Software Company

Jesse Yli-Huumo, Andrey Maglyas, and Kari Smolander

Lappeenranta University of Technology, Finland
{jesse.yli-huumo,andrey.maglyas,kari.smolander}@lut.fi

Abstract. Fierce competition in the software market forces companies to release their product under tough time constraints. The competition makes companies reactive and they need to release new versions often. To achieve this need for speed, companies take shortcuts to reach deadlines. These shortcuts and resulting omitted quality are called technical debt. We investigated one middle-size Finnish software company with two independent product lines and interviewed 12 persons in different positions to understand the causes and effects of technical debt. We were also interested in specific strategies and practices for managing technical debt. The results showed that technical debt is mostly formed as a result of intentional decisions made during the project to reach deadlines. Customer satisfaction was identified as the main reason for taking technical debt in short-term but it turned to economic consequences and quality issues in the longer perspective. Interestingly, neither of the product lines had any specific management plan for reducing technical debt but several practices have been identified.

Keywords: technical debt, software project management, case study, software company, software quality.

1 Introduction

The increased competition in the software market forces companies to think about their time-to-market strategy. Balancing the choice of releasing poor-quality software early or high-quality software late is challenging [1]. This leads companies to an awkward situation w here they have to decide what quality is omitted and what shortcuts in the development process they have to take. These shortcuts and omitted quality are called "technical debt."

The term technical debt was first introduced by Ward Cunningham [2]. According to Seaman et al. [3] compromises in software development are the reason for technical debt and they should be paid back to avoid decreasing maintainability and health of the system. Technical debt might also cause economic consequences in a software project [4]. McConnell divides technical debt into two different basic types [5]. Type I occurs unintentionally where a design approach turned out to be bad or a junior coder writes bad code. Type II is intentional where a company makes a strategic decision

A. Jedlitschka et al. (Eds.): PROFES 2014, LNCS 8892, pp. 93–107, 2014.

to incur technical debt. The technical debt metaphor appeals to both project managers and software development teams [6]. The goal of a software development team is to create quality software and embrace tools and techniques for it [6]. People responsible for the management of the project and the company also care about quality but are more focused on cost and schedule factors [6].

In this paper we define technical debt as a shortcut in a software development project to reach certain deadlines. We firmly believe that technical debt is not only related to technical decisions but also to management and business decisions. Moreover, the decision to have some technical debt can be conscious to deliver the product to the market faster.

This case study is conducted in one middle-size Finnish software company with two independent product lines. The objective of this case study is to understand the sources of technical debt in software projects. Our secondary objective is in identification of strategies and practices for the management of technical debt.

The rest of the paper is organized as follows. Chapter 2 shows the related work of this topic. Chapter 3 provides the research method used in this study. In Chapter 4 we introduce the results analyzed from the gathered data. Then, in Chapter 5 we discuss about the results and conclude the paper.

2 Related Research

In this chapter we identify the related research conducted on causes and effects of technical debt. We are also focusing on strategies and practices used for managing and reducing technical debt.

2.1 The Causes and Effects of Technical Debt

The study conducted by Lim et al. [7] shows that technical debt is not always a result of bad coding. It can also include intentional decisions to trade off competing concerns during business pressures. The study also identified that technical debt can be used in short-term to capture a market share and even to collect early customer feedback. In the long-term technical debt effects tended to be negative. These effects included increased complexity, poor performance, low maintainability and fragile code. This furthermore led to bad customer satisfaction and extra working hours. However, the authors also mentioned that there were cases where short-term benefits of technical debt clearly outweighed the future costs.

Klinger et al. [8] studied the causes of technical debt by interviewing four software architects at IBM. The study revealed causes for technical debt including pressure from stakeholders, technical communication gap between stakeholders and project team, decision making without quantification of possible impacts, and unintentional debt occurring from acquisitions, change of requirements, and changes in the market ecosystem.

Siebra et al. [9] analyzed an industrial project that lasted six years by analyzing emails, documents, CVS logs, code files and interviews with developers and project managers. The analysis revealed that technical debt was taken mainly with strategic decisions. They also found that the use of a unique specialist could lead the develop-

ment team to solutions that incur technical debt. The study also identified that the effects of technical debt can both increase and decrease the amount of working hours.

Zazworka et al. [10] studied the effects of god classes and design debt on software quality. God classes are examples of bad coding [11] and therefore include a possibility for refactoring. The results showed that god classes require more maintenance effort that include bug fixing and changes to software and are considered as a "cost" to software project.

Buschmann [12] explained three different cases of technical debt effects. In the first case technical debt in one platform started to grow so large that development, testing and maintenance costs started to increase dramatically. In the second case developers started to use technical debt to increase the development speed. This resulted to significant performance issues that turned later on to economic consequences. In the third case the software product had already incurred a huge amount of technical debt that led to increasing maintenance costs. However, management analyzed that reengineering the whole software would cost more than doing nothing. This resulted to situation where the management decided not to do anything for the technical debt because it was cheaper from the business point of view.

Guo et al. [13] studied the effects of technical debt by following one delayed maintenance task through a software project. The results showed that delaying the maintenance task would have almost tripled the costs if it had been done later.

Overall, the related research about the causes and effects show that technical debt is not always caused because of technical reasons. Studies [7][8][9] showed that technical debt can also be caused by intentional decisions that were related to business reasons.

The studies also show that taking technical debt may have short-term positive effects [7][9] such as the time-to-market benefit, but they will turn to economic consequences and quality issues in a long run if not paid back [7][9][10][12][13]. However, there are also situations where the short-term benefits overweigh the long-term costs [7][9].

We also noticed from the related research that technical debt is not just related to shortcuts in the coding phase. There are several different subcategories of technical debt mentioned in literature. We have gathered all of these technical debt subcategories in Figure 1.

Overall, the existing literature reveals large set of causes and effects incurring from technical debt, but lacks a clear mapping of relationships between different effects and causes.

2.2 Current Strategies and Practices of Technical Debt Management and Reduction

Lim et al. [7] found four different strategies for managing technical debt. The first strategy is to do nothing because technical debt might not be ever visible to the customer. The second strategy is the risk management approach to evaluate and prioritize technical debt's cost and value. The third strategy is to include different stakeholders

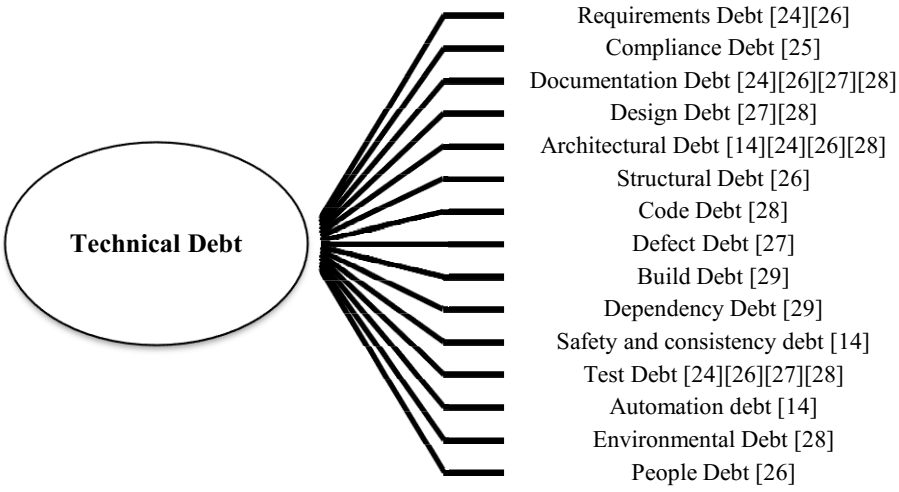

Fig. 1. Subcategories of technical debt

to technical debt decisions. The last strategy is to conduct audits with the development team to make technical debt visible.

Codabux & Williams [14] revealed practices such as refactoring, reengineering, and repackaging used for technical debt management.

The studies [15][16][17] propose the management of technical debt using the portfolio management. This approach is similar to the investment portfolio management. In the portfolio approach technical debt is collected to a "technical debt list" that is being used to pay the technical debt back based on its cost and value.

Krishna and Badu have also developed guides based on their own experience of technical debt management [18][19]. These guides are using different practices for minimizing technical debt. The practices include the basic steps that are focusing on improving refactoring, aspects of coding, continuous learning processes and teamwork. These practices were used in software projects and the results showed an improvement in the adaptation of new changes and better productivity in software project.

Overall, the existing literature is often concentrating on strategies and practices for reducing and preventing technical debt. As a result, the existing literature is lacking clear management approaches for controlling technical debt through the software development life cycle.

3 Research Methodology

Case study was selected as the research methodology for this study. Case studies have been around a long time and they account a large proportion of books and articles [20]. Case study is a method that involves an in-depth examination of a case [21]. Yin defines case study as an *'empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident'* [22]. In this study, we focus on technical debt

in software project. We focus more on technical debt causes and effects, rather than on the qualities of technical debt in source code and how to measure them. Therefore, we decided to use exploratory case study methodology with semi-structured interviews for data collection. The purpose of this study is to increase our knowledge of the relationship between technical debt causes, effects, and management. This case study consists of the following five steps [22]:

1. Designing the case study
2. Conducting case studies (stage 1): Preparing for data collection
3. Conducting case studies (stage 2): Collecting the evidence
4. Analyzing the case study evidence
5. Reporting the case study

3.1 Designing the Case Study

The strategy for this research was to find a suitable company to study technical debt and to arrange multiple interviews with people in different organizational positions (e.g. development, management, quality assurance). The reason for interviewing different organizational positions was to acquire information about technical debt from different viewpoints in software development projects. The research questions this study addresses are:

1. What are the causes and effects of technical debt?
2. What management and reduction strategies/practices are being used for technical debt?

3.2 Preparing for Data Collection

The selected company is a Finnish software company that offers SaaS business solutions for professional services automation and accounting. The company has three product lines that are managed and developed independently. This study includes two of the product lines that are referred to as Product Line A and Product Line B.

Table 1. The roles of the interviewees

ID	Product line	Role
A1	A	Software architect
A2	A	Software designer
A3	A	Project manager
A4	A	Software test engineer
A5	A	Production director
B1	B	Software architect
B2	B	Software developer
B3	B	Product line manager/Software test engineer
B4	B	Software architect
B5	B	Software developer
B6	B	UI designer

Product line A provides a financial management solution as a cloud service. The solution has more than 10 000 customers and it is currently the biggest financial management system provider in Finland. The product has been developed since 2004. The size of the development team in Product Line A is 18 persons. The development team is using agile methods and more specifically a Scrum-like approach.

Product line B is a SaaS-based project management solution for multi-organization projects. The solution is used by 1000 companies worldwide. Product Line B was founded 2004 and acquired by the parent company in 2010. The size of the development team in Product Line B is 13 persons. The development team is also using agile methodologies and a Scrum-like approach.

3.3 Collecting the Evidence

The interviews were conducted between February and March 2014. We had total of 11 interviews, of which five were from Product Line A and six from Product Line B. One of Product Line B interviews included an interview with two persons interviewed at the same time. The data collection started with a contact to the product line managers from both Product Line A & B. Both product managers agreed to have an interview and recommended persons from their own product line that might be suitable to interview about technical debt. We contacted all of the persons recommended and received positive response from each of them.

Table 1 presents the roles of the interviewees in this study. The interviewees were from different positions within the development team and we were able to discuss technical debt from various different viewpoints. We also tailored different questions depending on the role of an interviewee. For example if the interviewee was responsible for the management we focused our questions more on strategies and processes. All interviewees gave us a permission to record the session with a voice recorder. Seven interviews were conducted by one researcher in Finnish and the rest by two researchers in English. Later all interviews were transcribed and translated to English. Overall, the interviews lasted from 25 to 50 minutes, with the average of around 35 minutes.

3.4 Analyzing Case Study Evidence

The total amount of transcribed pages for analysis was 80. The collected data was analyzed with a tool designed for qualitative data analysis, Atlas.ti. The analysis of the transcribed data was performed with a similar method to open coding as in the grounded theory method [23].

4 Results

The main findings are presented in four subsections. The first subsection presents the causes related to technical debt. The second subsection examines strategies and practices for the management of technical debt. The third subsection focuses on the effects in short- and long term caused by technical debt. The final section presents improvements to product lines regarding technical debt.

4.1 Causes for Technical Debt

We asked from the interviewees what the causes for technical debt usually were. The most common cause mentioned was the lack of time given for the development. It was also added that the lack of time generates a lot of pressure to the development team, which ultimately leads to technical debt being taken.

"We have things that are almost well done but they are left undone properly because deadlines are coming and it is working like "ok" and we will just leave it like that." – B6.

We also identified that when the source code is getting more complex and bigger, it becomes also harder to change. This makes it easier and cheaper in the short run to just take technical debt and use patch code, instead of fixing the bigger problem. A software architect from Product Line A explained the situation where patch code was used to deal with the problem.

"We were working with our CRM product and our job was to refactor the data structure to this new type of data structure. This current data structure was used by almost every customer and it would have been really hard to change it. So we just made some script that syncs the data between these data structures and brings data to our CRM same way. This has generated a lot of problems, like in CRM the data is showing little bit differently even though it is the same data." – A1.

Another commonly mentioned reason for technical debt was the lack of knowledge. Sometimes it is just impossible to predict the future and some present decisions might incur technical debt later on. The interviewees also described different examples where lack of knowledge has caused technical debt to software product. One interviewee explained that lack of documentation in the source code causes technical debt because the code is then harder to understand and it takes more time to work with it. Also, if new coders do not have enough experience in coding with the company standards, they unintentionally incur technical debt because they produce different style of code compared to the company standards. The last example related to lack of knowledge described was about mistakes in specifications and requirements.

Another major aspect we were interested in at the causes of technical debt was the effect of the business decisions. We asked interviewees about the effects of business decisions to technical debt by causing pressures to the development team. A software architect in Product Line B thought that business decisions do have effect to the amount of technical debt.

"Previous years business decisions have affected us a lot. Reason for this that we have a release every two months and when we have a bigger release we just do not have enough time to do it. This means that we are in a hurry at the end of release and we have to just implement some faster solution. I think that problem is that management is not willing to see what people are actually doing during work day and how much time is doing something actually taking. This leads to problems in distribution of resources and I think that has been our problem these days." –B4.

A project manager of Product Line A also thought that business decisions are generating pressure to the development team, but he also added that it is not completely the decision of business managers to make deadlines for features.

"Well of course there is pressure coming, but what is the weight of it is case-by-case. At some point if we have promised something to our customer, we try to give deadlines so that there should not be any pressure coming. Of course deadlines are changing and there are situations where we have to reach fast and everything else is postponed, but if we were in a perfect world, this is how we would deliver." – A3.

We also asked about the communication between the development team and the business management related to decision making on technical debt. Interviewees from both product lines thought that they have a good communication structure where project managers act as good filters between business managers and development team for the decisions on technical debt. Some interviewees also added that they consider this communication structure to be much better compared to companies they have previously worked in. The reason for this was that the previous places were big companies, where getting an opinion about technical debt was more challenging due to more complex and larger communication structure.

We were also interested in whether business people usually listen to the technical opinions of the development team. Most of the interviewees thought that it is difficult to express technical opinions to business managers and to get more development time. However, a project manager of Product Line A expressed that the development team often mentions situations related to technical debt.

"Well we try internally say that if we don't have capacity for some feature to make it 100% solution, it will be told usually at most of the cases. We will picture it that we don't have time to make this and this feature completely. So they understand us quite well." – A3.

We asked could one of the causes for technical debt be the lack of technical knowledge of business managers that might drive not to do the best possible solution. The interviewees thought that it might be the case but did not see it as a problem because project managers usually can express their opinions to these kinds of situations, which limits the amount of bad decisions.

Also, one interesting fact we noticed during interviews was that the examples given by interviewees were not just related to the implementation phase. Interviewees also described how shortcuts were taken for example in testing, architecture and requirements phases of the software life cycle.

4.2 Management and Reduction Strategies for Technical Debt

The goal was to find out if the product lines A & B have any clear strategies for managing and reducing technical debt. Neither of the product lines had any specific approach for managing technical debt. However, we were able to identify some practices that were used to report technical debt.

"Well for bigger things we do keep a backlog. If some feature is dropped out we do have a backlog for it. If I am thinking myself at coding and I take some shortcut

somewhere, so do I mark-up it somewhere, not really. But if we are dropping something or finding some things to take, we write them up." – A1.

One interviewee from Product Line B mentioned that they are using a software called JIRA to store their actions during development.

"Well there is that if we decide that we are going to make it better later, we have this software called JIRA, where we have all the tickets about things we have done. So at least it will be stored in there, other thing is that when it will be fixed." – B6.

Even though neither of the product lines used any specific approach to manage technical debt we managed to identify practices that were being used to reduce and prevent technical debt. Both of the product lines for example have a specific bug fixing day each week to reduce errors and bugs.

We identified that both product lines were using refactoring to reduce technical debt. However, they did not have any specific refactoring schedule. Refactoring was considered as a part of normal job and was done during the implementation.

Both product lines were using coding standards/guides to prevent and reduce technical debt coming from bad coding and to increase the consistency of the source code.

"We have guides, but they are still missing a lot of stuff. We are combining them all the time and adding stuff. But at the moment they are mostly describing naming the code, architectural stuff. We have different guides for different projects, like different guides for APIs compared to normal software. But at least we have something." –B4.

For the situation where bad solutions or inconsistency still occurs even after coding standards/guides, both companies had code reviews to check all produced code. This served as the last checkpoint where technical debt can be seen before a release.

4.3 Short-Term and Long-Term Effects of Technical Debt

The analysis revealed several effects how technical debt can affect a software development project in a short-term. First, technical debt, or a shortcut, is used to save development time and deliver a solution faster to the customer. The production director of Product Line A explained a situation from the early stages of the product where taking technical debt saved the company.

"Well I think that the best example is when we started the company in 2001. We were quite out of money, so we were leveraged with debt quite heavily and we were in a hurry to get something done for the next presentation for pilot companies and investors. So we created a lot of stuff that looked it worked, but in reality it might not even work. Time-to-market was so important for the existence of a company, so we did everything we could just to get stuff out to convince that this is a viable solution." – A5.

Second, the customer satisfaction can be increased by delivering the solution faster but it also increases the technical debt. As customers are more interested in getting the product on time rather than its technical details of implementation, they do not care

about technical debt as long as it does not directly affect the product quality. One interviewee also mentioned that taking technical debt doesn't feel good to take from developer's point of view because they know that the solution is not the best one and they might have to fix it later.

We were also able to identify long-term effects occurring from technical debt. The interviewees from both product lines explained that if technical debt is not being managed and reduced, it will have some serious effects in the long-term. A software architect from Product Line B explained a situation where technical debt effects have caused problems to the product line.

"We have things like calendar synchronization, which we have done years ago and there is a lot of bugs and errors in the functionality and problems in implementation. These kinds of things have generated us a lot of bugs in the long-term and lots of repairing. Repairing has been done by putting patches somewhere and not by creating a totally new base for it. For example we have had this calendar synchronization for two years and I believe we have used hundreds of hours for fixing bugs after its release. We should take plenty of time and look at the big picture." – B4.

The common effect mentioned by interviewees was more working hours spent for recoding and fixing errors/bugs of solutions made with technical debt. The interviewees felt that the solutions built on top of the already bad solutions were basically already implemented wrong and fixing with patch code is just postponing the issue. They explained that technical debt lowers the quality and performance of the product in the long run and it ultimately leads to a decrease in customer satisfaction.

4.4 Future Improvements for Dealing with Technical Debt

We were able to some identify possible improvements related to dealing with technical debt in both product lines. Neither of the product lines had any specific approach for dealing with technical debt management and reduction. Interestingly, majority of the interviewees thought that they would need improvement for that. We asked the interviewees about the backlog type management where technical debt is being managed and reduced through listing all the shortcuts to a backlog and starting the reduction from there.

"Well some feature control would be good, where you can see that what has been done in a short way and other stuff. So some kind of feature management system. Usually companies have these, but smaller the company less systems." – A2.

We were also able to identify practices related to refactoring, coding standards/guides and code reviews that were used to reduce technical debt. We noticed that the continuous delivery of new features is taking time away from refactoring. This might lead into a situation where technical debt stays in the software because the development team has to continuously implement new features. The risk is that if this debt is forgotten it might cause problems in the long-run. We think that having some refactoring time after every release to reduce the technical debt might be a good

solution, instead of just moving to the new features of next release. Also one possible solution could be to assign for example two developers to do only refactoring. Project manager of Product Line A mentioned that they are trying to improve the estimation of deadlines and include also technical debt in this.

"Lately we have been trying to include, if we have some gap in some feature and if we have to do something to that feature and we know that there is something existing in that feature, that we make some time to fix it correctly." –A3.

We also identified practices in coding standards/guides and code reviews. We noticed that coding standards/guides are not used by everyone and they are also not updated. The lack of coding standards/guides has an effect on the consistency of the source code and especially junior coders are more exposed to write bad code that is considered as unintentional debt. Also, we identified that in the other product line the code reviews were not conducted on a regular basis, but only after a release. With code reviews it is possible to interrupt these bad solutions before they are included in the release of the software. However, we noticed that both product lines have acknowledged these problems and are currently improving them by updating the coding standards/guides and increasing code reviews. We think that improving these two aspects will have an impact on preventing unintentional technical debt.

4.5 Summary of the Findings

In Table 2 we summarize the results of this case study. We were able to identify several different causes for technical debt. These can be further divided into technical debt that is a caused with intentional decisions and technical debt is incurring unintentionally. We were also able to identify several short- and long-term effects of technical debt to a software project. As expected, the effects of technical debt seemed to be positive in a short-term, but turned negative in a long-term. Although we did not find any specific approach for managing technical debt, we were able to identify some practices for reducing technical debt.

5 Discussion and Conclusions

With this study we were able to identify empirical evidence from the relationship between technical debt causes, effects, and management. The results from both of the product lines are similar and clearly show that technical debt is appearing in both of them. McConnell [5] defined that technical debt can be divided into two different main types intentional and unintentional debt. The examples given by interviewees also show that technical debt is occurring in product either unintentionally or with intentional decisions. Based on these findings we agree with McConnell [5] that technical debt can be divided into these two main types. Moreover, based on our observations, technical debt does not seem to be only related to coding, where a coder takes a

Table 2. Summary of the findings

RQ1: What are the causes and effects of technical debt?	
Intentional causes of technical debt	• Lack of time given for development • Pressure to the development team • Complexity of the source code • Business decisions - Lack of technical knowledge - Communication challenges
Unintentional causes of technical debt	• Lack of coding standards and guides • Junior coders • Lack of knowledge about future changes • Lack of documentation
Short-term effects of technical debt	• Time-to-market benefit • Increased customer satisfaction
Long-term effects of technical debt	• Extra working hours • Errors and bugs • Customer unsatisfaction • Complexity of the source code
RQ2: What management and reduction strategies/practices are being used for technical debt?	
Practices for reducing and preventing technical debt	• Refactoring • Bug fixing days • Code reviews • Coding standards and guides • Communication structure between business management and development team

shortcut in the source code. Instead, the results show that similar effects to take shortcuts were happening in different phases of the software development life cycle. Our observations suggest that similar phenomenon to take shortcuts can happen also in requirements, architecture and testing phases. We argue that technical debt should not be limited to shortcuts in source code only, but it should also include shortcuts in other phases of the software life cycle as well. Dividing technical debt into more specific subcategories may bring more clarity to the concept of technical debt over the whole software development life cycle.

The first research question was related to the causes of technical debt in the software development life cycle. The results suggest that technical debt is not necessarily caused by a single specific reason. The effect of the lack of time for the development was identified as the primary reason for technical debt in a software project. We also identified that common causes for the lack of time and pressure for the development team include business decisions. These findings are similar to other studies [7][8][9] where the researchers identified that taking technical debt is also caused by intentional decisions. We believe that the lack of time for development ultimately comes from business realities that set up the deadlines for project based on customer needs and current market situation. This makes the development team to take shortcuts to meet deadlines. However we also noticed that the both product lines had built a communication structure between the business and development departments that increased the

capability of the development team to express their opinions on technical debt decisions. Klinger et al. [8] found in their study at IBM that the cause for technical debt is the technical communication gap between business managers and the development team. Based on these observations we think that in small and middle sized companies technical debt decisions might be easier to deal with when compared to large companies where the communication structure is more complex. Another thing we also observed in these cases was that unintentional debt was mainly caused by the lack of knowledge about future and lack of coding standards/guides that will especially affect junior coders.

The second part of the first research question was the short- and long term effects of technical debt in the software development life cycle. Other studies [7][9] argue that the good thing about technical debt is the short-term effect of time-to-market. We were able to identify similar situations, where technical debt was used to get the solution out faster. In the long-term technical debt tends to have more negative effects [7][9][10][12][13]. Our observations also revealed situations where technical debt in the long-term started to generate extra working hours and errors/bugs. The effects of technical debt are mostly positive the moment you take them but might turn into problems later if they are not paid back. This could be the reason why business people think that technical debt is something that is really easy to take to meet the deadlines and just fix it later.

The second research question focused on strategies of managing and reducing technical debt. Neither of the product lines had any specific management strategy for technical debt. However both product lines were using some practices to collect the technical debt items to a backlog, but they did not have any reduction strategy for them. A portfolio management strategy proposed in other studies [15][16][17], where technical debt is stored to backlog and the development team can use that for management and reduction, would be a good option to technical debt management. This kind of a backlog strategy might be beneficial to product lines in a long-run when older technical debt is traceable, instead of forgotten. Even though neither of the product lines had any clear strategy for managing technical debt we identified several practices used for reducing it. The practices included refactoring, coding standards and guides, code reviews, and specific bug fixing days. Similar practices have been proposed also in other studies [14][18][19]. We believe that all these practices can reduce and prevent the amount of technical debt and also increase the overall quality of the product.

In conclusion, technical debt is something that companies are unable to avoid during their software development projects. In this case study technical debt was formed as a result of different management level decisions that were made during the project to reach deadlines or unknowingly due to the lack of knowledge. However, technical debt is not always a bad thing to take. Companies can use technical debt as a powerful tool to reach their customers faster to gain an edge over the competition in the market. Nevertheless, if technical debt is not paid back in time, it might generate economic consequences and quality issues to the software. To use technical debt correctly companies need to create a management plan including practices that decrease technical debt.

Acknowledgement. The authors would like to thank the company and their employees for participating to this research. This research has been carried out in Digile Need for Speed program, and it has been partially funded by Tekes (the Finnish Funding Agency for Technology and Innovation).

References

1. Van de Laar, P., Punter, T. (eds.): Views on Evolvability of Embedded Systems. Springer, Dordrecht (2011)
2. Cunningham, W.: The WyCash Portfolio Management System. In: Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA 1992, pp. 29–30. ACM, New York (1992), http://dl.acm.org/citation.cfm?id=157715 (accessed: March 25, 2014)
3. Seaman, C., Guo, Y., Zazworka, N., Shull, F., Izurieta, C., Cai, Y., Vetro, A.: Using technical debt data in decision making: Potential decision approaches. In: 2012 Third International Workshop on Managing Technical Debt (MTD), pp. 45–48 (2012)
4. Zazworka, N., Vetro, A., Izurieta, C., Wong, S., Cai, Y., Seaman, C., Shull, F.: Comparing four approaches for technical debt identification. Software Quality Journal, 1–24 (2013)
5. McConnell, S.: Technical Debt-10x Software Development | Construx (November 01, 2007), http://www.construx.com/10x_Software_Development/Technical_Debt/ (accessed: March 25, 2014)
6. Eisenberg, R.J.: A Threshold Based Approach to Technical Debt. SIGSOFT Software Engineering Notes 37(2), 1–6 (2012)
7. Lim, E., Taksande, N., Seaman, C.: A Balancing Act: What Software Practitioners Have to Say about Technical Debt. IEEE Software 29(6), 22–27 (2012)
8. Klinger, T., Tarr, P., Wagstrom, P., Williams, C.: An Enterprise Perspective on Technical Debt. In: Proceedings of the 2nd Workshop on Managing Technical Debt, New York, NY, USA, pp. 35–38 (2011)
9. Siebra, C.S.A., Tonin, G.S., Silva, F.Q.B., Oliveira, R.G., Junior, A.L.O.C., Miranda, R.C.G., Santos, A.L.M.: Managing Technical Debt in Practice: An Industrial Report. In: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, New York, NY, USA, pp. 247–250 (2012)
10. Zazworka, N., Shaw, M.A., Shull, F., Seaman, C.: Investigating the Impact of Design Debt on Software Quality. In: Proceedings of the 2nd Workshop on Managing Technical Debt, New York, NY, USA, pp. 17–23 (2011)
11. Vaucher, S., Khomh, F., Moha, N., Guéhéneuc, Y.: Tracking Design Smells: Lessons from a Study of God Classes. In: 16th Working Conference on Reverse Engineering, WCRE 2009, pp. 145–154 (2009)
12. Buschmann, F.: To Pay or Not to Pay Technical Debt. IEEE Software 28(6), 29–31 (2011)
13. Guo, Y., Seaman, C., Gomes, R., Cavalcanti, A., Tonin, G., da Silva, F.Q.B., Santos, A.L.M., Siebra, C.: Tracking technical debt - An exploratory case study. In: 2011 27th IEEE International Conference on Software Maintenance (ICSM), pp. 528–531 (2011)
14. Codabux, Z., Williams, B.: Managing technical debt: An industrial case study. In: 2013 4th International Workshop on Managing Technical Debt (MTD), pp. 8–15 (2013)
15. Power, K.: Understanding the impact of technical debt on the capacity and velocity of teams and organizations: Viewing team and organization capacity as a portfolio of real options. In: 2013 4th International Workshop on Managing Technical Debt (MTD), pp. 28–31 (2013)

16. Guo, Y., Seaman, C.: A Portfolio Approach to Technical Debt Management. In: Proceedings of the 2nd Workshop on Managing Technical Debt, New York, NY, USA, pp. 31–34 (2011)
17. Zazworka, N., Seaman, C., Shull, F.: Prioritizing Design Debt Investment Opportunities. In: Proceedings of the 2nd Workshop on Managing Technical Debt, New York, NY, USA, pp. 39–42 (2011)
18. Krishna, V., Basu, A.: Minimizing Technical Debt: Developer's viewpoint. In: International Conference on Software Engineering and Mobile Application Modelling and Development (ICSEMA 2012), pp. 1–5 (2012)
19. Krishna, V., Basu, A.: Software Engineering Practices for Minimizing Technical Debt. presented at the SERP 2013 The, International Conference on Software Engineering Research and Practice (2013)
20. Denzin, N.K., Lincoln, Y.S.: The SAGE Handbook of Qualitative Research, 4th edn. Sage Publications
21. Verner, J.M., Sampson, J., Tosic, V., Bakar, N.A.A., Kitchenham, B.A.: Guidelines for industrially-based multiple case studies in software engineering. In: Third International Conference on Research Challenges in Information Science, RCIS 2009, pp. 313–324 (2009)
22. Yin, R.K.: Case study research: design and methods. Sage Publications, Thousand Oaks (2003)
23. Strauss, A., Corbin, J.M.: Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. SAGE Publications (1998)
24. Ojameruaye, B., Bahsoon, R.: Systematic Elaboration of Compliance Requirements Using Compliance Debt and Portfolio Theory. In: Salinesi, C., van de Weerd, I. (eds.) REFSQ 2014. LNCS, vol. 8396, pp. 152–167. Springer, Heidelberg (2014)
25. Kruchten, P., Nord, R.L., Ozkaya, I.: Technical Debt: From Metaphor to Theory and Practice. IEEE Software 29(6), 18–21 (2012)
26. Zazworka, N., Spínola, R.O., Vetro', A., Shull, F., Seaman, C.: A Case Study on Effectively Identifying Technical Debt. In: Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering, New York, NY, USA, pp. 42–47 (2013)
27. Tom, E., Aurum, A., Vidgen, R.: An exploration of technical debt. Journal of Systems and Software 86(6), 1498–1516 (2013)
28. Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C., Sullivan, K., Zazworka, N.: Managing Technical Debt in Software-reliant Systems. In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, New York, NY, USA, pp. 47–52 (2010)
29. Morgenthaler, J.D., Gridnev, M., Sauciuc, R., Bhansali, S.: Searching for build debt: Experiences managing technical debt at Google. In: 2012 Third International Workshop on Managing Technical Debt (MTD), pp. 1–6 (2012)