The effects of a model for identifying and managing Technical Debt in software projects

Rogerio C. Pires¹, Graziela Tonin², Alfredo Goldman², Renan Oliveira²

¹Technological Research Institute of the State of São Paulo (IPT)
²Institute of Mathematics, Statistics and Computer Science – University of São Paulo (USP)

Rua do Matão, 1010 Cidade Universitária – São Paulo – SP – Brazil
{grazzi, gold}@ime.usp.br, {rogerio.chaves, renandemelo}@gmail.com

Abstract. Technical Debt is a metaphor in the software engineering process [Cunningham 1992] that compares the negligence of quality to as acquiring a debt. In this context, we performed a multiple case study conducted on three software projects in a large Internet Company trying address the following question: "How to effectively manage Technical Debt in an agile project?". We conducted the study by proposing a model for managing Technical Debt and applying a qualitative survey to collect team member's perceptions - further coded and analyzed. As an outcome of this study, a model for managing Technical Debt in agile teams emerged from the experience, which we provide for agile practitioners interested on additional guidance in this topic.

1. Introduction

Events in which quality in software engineering is neglected often occur in an ordinary software development project. Such events might happen for several reasons, such as a tight budget, lack of time and/or resources, and the need to deliver features fast in order to maximize the return of the investment (Return of Investment – ROI). In this sort of scenario, immature artifacts are often inserted into the software developed by team members. This phenomena is informally known by software practitioners as Technical Debt - a metaphor presented for the first time by Ward Cunningham [Cunningham 1992] while describing his experience in the conference Object-Oriented Programming, Systems, Languages & Applications (OOPSLA) 1992:

“...Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite... The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise...” (Cunningham, 1992)

The metaphor of Technical Debt is an analogy for financial debt. It claims that when the quality of the software is compromised in detriment of any other benefit, it is the same as if you were in debt. Interests are added to the debt in a way of implementing future changes in the system. If the debt is not controlled, part of future development of the team might be compromised in paying interests.
Recently, the concept of Technical Debt has gained popularity among both in the industry and the academia. Sometimes, due to a strategic decision during the process of development of a system, the quality of the code is mainly compromised given constraints imposed on the process such as restrictions of time or other resources. Moreover, developers usually have the option to compromise the quality of the software in order to meet urgent demands in another dimension of the system - time to market, for example. Such an event is sometimes compared to acquiring a debt that needs to be paid in the future. In this analogy, this event implies on interests that can go up in terms of time, efforts or extra costs of future changes [Guo 2009].

Technical Debt has been a topic of many discussions in blogs, forums, lectures and events. On the one hand, there are many professionals in the field of the development trying to figure an effective way out to apply this concept in their companies. The academia is also interested in the topic. Some research has been done in the area intending to formalize a theory on Technical Debt [Brown et al 2010] while others aim to discover helpful ways to identify and manage it [Guo 2009] [Guo and Seaman 2011] [Guo at all 2012] [Tonin 2011]. However, there is still a paucity of empirical evidence on how to use, effectively, the concept of Technical Debt, while it has been responsible for great losses in the economy. For example, a recent study points out that approximately US$500 billion were spent in 2010 with Technical Debt and it is estimated that until 2015 the expenses will reach US$1 trillion [Gartner 2012].

In turn, some works in the software community report that companies have adopted Agile Methods as a way to manage software development in the last ten years. The benefits of the agile development paradigm have been proved meaningful, while there is some evidence of improvements in important areas such as productivity, the stress and customer satisfaction [VersionOne 2011] [Williams 2010]. Research have pointed out that agility is a valuable paradigm for organizational success, while studies from MIT show that Agile companies increase the income 30% faster and generate 37% more profit than the Non-Agile [Highsmith 2011]. However, according to Highsmith [2011], quality, or the lack of it, is still the main point for the effective agile.

In this paper, we describe a study applied in a few agile projects in a large Brazilian Internet company aiming to explore the possibilities for managing Technical Debt in an agile environment. It is organized as follows. In Section 2, we describe the design of our research. In Section 3 we describe the data collected during the study, while we report our results in Section 4 and present our conclusions in Section 5. Additionally, we describe threats to the validity of this study in Section 6 and work to be further addressed in this topic in Section 7.

2. Research Design

Given the dearth of empirical studies addressing the topic of managing Technical Debt, specifically in agile software projects, we performed a study in order to better understand the effects of applying a Technical Debt management model for agile teams. After performing an initial literature review, we could not find any empirical study addressing the following research question, which we propose to help in answering:

RQ. How to effectively manage Technical Debt in an agile project?
2.1. Research Method

Given the contextual nature of both topics of agile development and Technical Debt management, we decided to address the research question by performing a multiple case study. As described by Yin, case study research is known by allowing the investigation of “a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident” [Yin, 1989].

In this multiple case study, we selected three different agile projects in the same department of a Internet Software Engineering Company in order to apply the same research design protocol. We followed the same steps for all projects in order to deeply understand their specific contextual characteristics and how they affect management of Technical Debt. Another advantage of multiple case studies is providing to the researcher a higher level of generality on the study results. Moreover, both similarities and differences of each case become evident as one compares collected data from different cases.

We divided the research protocol performed in each case as a series of steps, which included acknowledging each project context characteristics, unifying the concept of Technical Debt among the team members, identifying specificities each process by participating in some traditional ceremonies in agile development, and proposing a framework for managing Technical Debt for each environment. At end, we collected qualitative data throughout a survey applied to all team members. The overall research design is available in Figure 1 and is described in the following subsections.

![Figure 1. Research Design](image)

2.2. Context

This research was carried in a large IT company with many products focused in the Internet industry. The company was founded in 1996, originally as an Internet service provider focused in dialed service content. Thereafter, it started to diversify its range of products and services while began to evolve its technologies and practices usually adopted in the Internet industry. The company software engineering staff is currently composed by more than 150 developers, divided into approximately 30 teams. We conducted the study on three selected projects from the same department – internally called as Research and Development (R&D) – which is responsible for the conception, development, and maintenance of several software products and services provided by the company's Internet portal. All projects were performed by software teams that claimed to base their development practices on agile development – having a higher focus on Scrum.
a) Monitoring and Automation System project

This project has started in 2008, with the purpose of creating a product to monitor and automatize the products developed by the company. It works as an integrated platform of all company products. The product development tools are basically Java, Perl, Puppet, Tomcat, Webservice Jet, System of Queue control ActiveMQ, and Nagios. They informed that it is not a system, but a software platform that could be integrated in all company systems.

b) Host platform project

The project has also started in 2008. In the occasion, a plan of the system was done. The system should be developed in three or four months, due to the company strategies in launching in the market of host sales. The current project technical leader begun in the company in 2010, and has taken a product in production with the modules in the initial version, totally developed and based on a monolithic system with a large technical dependence of other systems. The development process, until 2010, was not integrated to the process based on the Scrum defined by the company. Because of this, they followed the development without adopting the patterns and process existing so far. The product was created in a nonstandard way, without the operation process and the software development good practices. From 2010, it begun to standardize the process adopted for the company operation, unifying the operational team and standardizing the code and components of the system. In this process, it has generated a huge rework to rebuild the components of the system. “I don’t have doubts that all this work was a payment of Technical Debt”, says the responsible for the product. And to him, these Technical Debts might be necessary, because if the company has not had a fast launching of the product in the occasion, maybe it could not have got positioned in the market in such a solid format as it has been now.

c) Domain platform project

The project started in April 2008 by the company initiative to enter in the market to sell web domains. A third party company conducted operations and the original product development took 6 months until the first deployment. After the delivery of the first software release, the company created a development team for product evolution. It is based on Java, PHP and Perl technologies. Since the first release, changes in the software consisted only in adaptations and layout settings improvements. These adaptations were being added to the product without planning, making the software maintenance quite complex.

2.3. Step 1. Contextualization interviews

In order to collect contextual characteristics of the project, we performed a series of semi-structured interviews, one for each project by team leader, aiming to take some project characteristics into account such as the history of each project and previous knowledge/decisions related to Technical Debt already taken by each the team. In this step, we used the following interview protocol:

1. How many professionals work in your team and how many years of experience each one has?
2. Could you describe the professional profile of each team member?
3. How much time is this team working together? How much time has your process being applied in the company?
4. When did the project start?
5. Is it a project for the development of a product not yet released or it is a maintenance project?
6. What is the average time of each Sprint?
7. Describe the involvement of the Product Owner in the software development process.
8. How the team prioritizes tasks/histories?
9. How is quality handled throughout the product development life cycle?
10. Do you consider your team has previous knowledge in the concept of Technical Debt? If it does, how it is managed throughout the project life cycle?

2.4. Step 2. Concept alignment
For a second step, we performed interventions on the software development teams in order to align the concept of Technical Debt and explain the purposes of the study. For each reason, we performed a specific presentation to the teams. First, in order to align the concept of Technical Debt, we performed a 45 minutes presentation based on a material provided by a specialist in Technical Debt – Dr. Carolyn Seaman. Second, we performed a 30 minutes presentation about the study in order to answer questions and promote research collaboration with the team members of each project.

2.5. Step 3. Identifying Specific Processes
In order to know about the differences in the process of each project, we participated in planning meetings of each team, in each project. We participated for three Sprints in each project. We decided to use the Spring planning meeting since it is traditionally an environment in which all team members discuss next steps of evolution. During these meetings, we could perceive aspects such as tools used by the team throughout the Sprints and how software quality is addressed or not by team members. By participating in such meetings, we were able to collect different opinions from team members about the process used by the team. At the same time, in order to extend the conceptualization of Technical Debt, we promoted discussions during the meetings on how this concept might be related to tasks to perform during the next Sprint.

As a forth step, we proposed a framework for managing Technical Debt based on the feedback received by all three projects. This proposal was also based on works by Seaman which argues that Technical Debt management should embrace three major steps: 1. Identifying Technical Debt; 2. Estimating Technical Debt; and 3. Taking a decision how to act in each case.

2.7. Step 5. Data collection
In order to help in answering the research question, we collected data from each team, further analyzed using qualitative coding procedures. We collected this data by applying a survey to team members after a few Sprints using the proposed model to manage Technical Debt. We designed the survey trying to collect the perception of team members on concept
of Technical Debt, the effectiveness of the proposed model and possible effects regarding quality and productivity. The survey protocol is available as follows:

1. Did the concept of Technical Debt become clear for you? Do you think that your team may be able to produce software with fewer defects with it?
2. By identifying Technical Debt, did the team manage to put more effort to develop in new features instead of putting in paying debts?
3. Do you believe that using tools (Jira - in this case) makes it easier to manage Technical Debt?
4. Having the proposed model in hands, do you believe that from now on your team will produce software with less defects and higher quality?

By applying the survey, we managed to gather data in order to understand the effects of deploying this model for managing Technical Debt. To analyze it, we used open and axial coding procedures described by Strauss & Corbin [1998]. Their proposal for these two qualitative coding procedures can be briefly summarized as:

- **Open coding**: The researcher passes by all data, identifying the core meaning of each part of data and applies a name – a code – to each part. Thereafter, the researcher groups codes into concepts and raise up larger groups – categories – to represent the data in a higher level of abstraction.
- **Axial coding**: After identifying categories, the researcher finds relationships among them by returning to raw data and identifying such links.
- **Selective coding**: A central category is elected while related categories are emphasized in the study.

We chose NVivo (http://www.qsrinternational.com/products_previous_products_nvivo8.aspx), as a tool for helping in the coding procedures while it is often used by social researchers for coding and analyzing qualitative data.

### 3. Collected Data

After the step of collecting information on how each team works, we made suggestions to the teams on how to highlight and control Technical Debt in the sprints.

Carolyn Seaman [2012] set up a framework for Technical Debt management in which the identification of the debt should be performed in three steps: identification of the debt, estimation debt and decision making on what to do.

![Figure 2. Technical Debt management framework](image-url)
Based on this framework, a proposal was made for the teams. This proposal was presented in a meeting held with the responsible of the three teams and we proposed some modifications in the Jira tool and some forms of control. This proposal was suggested to highlight and control the Technical Debt in the product backlog. Some parameterizations proposed, in the Jira, are fundamental to the proposal effectiveness:

- **Story type**: Every story created in a tool has type identification. Existing types, so far, were improvement and bug fix. One more type was added: Technical Debt.
- **When a Technical Debt story is selected**, some options appear in the type of debt (architecture, bug, testing and documentation).
- **Also**, some fields have to be completed to insert the value of the interest of the Technical Debt, in case it is not paid and another to define the probability of occurrence (high, medium and low).

Below there is the model defined in Jira for the Technical Debt control:

![Create Issue](image)

**Figure 3. Creating a Story using Issue Tracking Tool**

- **Technical Debt Story**: When we have a story in the sprint planning specifically about Technical Debt, include also its category;
- **Technical Debt Type**: When selected Technical Debt, include the type of debt (Architecture, bug, tests and documentation);
- **The amount of Interest to each debt**: how much it would cost to pay it now.
- **The probability of Interest (High, Medium and Low)**: this metric is whether the team or developer believes it can happen again.

In the moment you have a Technical Debt mapped/identified, it can be estimated both in the sprint planning’s and estimation meetings, along with other team stories, and properly prioritized by the Product Owner, according to their criticality and project development impact.
After this identification, it will be possible to collect metrics regarding Technical Debt incidence in the team development process, how it affects them and also with relation to development costs for the company (based on the allocation of the team in the development process or Technical Debt work).

Another point would be included in a whiteboard, where the teams controlling the daily meetings stories, an area for the team members include Technical Debts found during the sprint. In team meetings with the PO, the Scrum Master takes these suggestions to be inserted into Jira and evaluated by the team and PO.

These controls aims to highlight for the team and the Product Owner, the occurrence of Technical Debt, show it at sprint planning meetings and let clear their incidence and impact in the development process.

4. Results

The resulting model is shown in Figure 4.

![Figure 4. Model to Identify and Monitor Technical Debt](image)

When the team start to identify Technical Debt, it is possible to use the tools of project management to register the list of Technical Debt, given it is only necessary to create a template with fields that must be filled on a first in which the team will register the debt. Then we suggested that in some Sprints the team talk about the list of Technical Debts and decide whether it is necessary to insert a story to pay off some debts techniques. If the team decide paid, then select the debts to be paid and place them on the Scrum board so that any team member to pay.
4.1 Benefits for the team

The study identified some benefits generated for the team after the implementation of the model. Below we describe the results after each step of the data analysis.

In the open coding analysis, the codes were defined and some examples of these codes are listed in Table 1.

<table>
<thead>
<tr>
<th>Conceptual category</th>
<th>Ocurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical debt management</td>
<td>11</td>
</tr>
<tr>
<td>quality enhancement</td>
<td>10</td>
</tr>
<tr>
<td>Technical debt definition</td>
<td>10</td>
</tr>
<tr>
<td>history documentation</td>
<td>10</td>
</tr>
<tr>
<td>Positive points with technical debt management</td>
<td>9</td>
</tr>
<tr>
<td>Technical debt visibility</td>
<td>8</td>
</tr>
<tr>
<td>Negative points with technical debt management</td>
<td>8</td>
</tr>
<tr>
<td>Technical debt payment</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 1. Preliminary codes

The code Technical Debt management had more evidence in the survey answered by team members, highlighting the relevance of the topic: “…Once recognized and analyzed the Technical Debt stories, it is possible to make explicit software problems and identify critical points in the system quality process…”.

Others codes very mentioned were quality enhancement and Technical Debt definition. As reported by team members, managing Technical Debt makes the teams produce software with best quality and team members assimilated the concept and benefits of managing Technical Debt.

In the second step, the codes were rearranged in a level of greater abstraction. Thus, new combinations are again set to form the categories that, in turn, are organized. Below in Figure 5 we represent the categories:

![Figure 5. Relationship among categories](image-url)
The model implemented, was based on the tools and processes existing in the company. After implementation, the team members now have greater awareness to incur new Technical Debts and greater visibility over it: "... Now the team exposes to PO the Technical Debt existing in the software and these stories are performed by team with the stories of improvement and evolution of the product...". The model facilitates the management of the stories with others that were already treated by the team, providing the members measured the cost of payment of Technical Debt and interest payments associated Technical Debt.

The stories management model led teams to better planning, a proper documentation with a better categorization of stories that are Technical Debt and estimate the time and priority in payment ".... Stories for Technical Debt solutions were incurred and prioritized according to the cost of your payment, with the stories of product improvements...". This documentation allows better control and better negotiation with the Product Owner to prioritize Technical Debt stories "... Now the team exposes to Product Owner the Technical Debt that exist in the software and these stories are performed by the team along with the stories of improvement and product evolution...".

With Technical Debt control the teams can better plan for future refactoring of the product without acquiring new debt. Managing Technical Debt brings an improvement in software quality: "... You can make explicit points of software problem and identify critical points in the quality of an informed way system, helping to prioritize and correct problems, which optimizes search for a better quality in the system...”

In the last data analysis the categories were refined and integrated unveiling a category deemed as central, pervading all the others in this case, the objective is not to generate a theory, but instead to identify the main categories, the principal influence to identify Technical Debt. The mains result is describe in Figure 6:

![Figure 6. Selected categories](image)

The categories of Technical Debt Management, visibility of Technical Debt, planning, negotiation, documentation, simplicity, control and software quality are the major influences on the teams in the model implemented. Each category is part of the results, although none of them shows completely. For this reason, abstract categories, a conceptual idea under which all categories are included are necessary. For this reason, we conclude that the resulting main category is "Technical Debt Management". It was clear to all teams that identify and manage Technical Debt brings most visibility in the development process.
These Technical Debts documented in Jira tool, are transformed into stories and prioritized along with other product enhancements stories in a negotiation process with the Product Owner, making the product more quality. All this management model and stories control, was defined based on a simple way in which the processes and systems preexists were maintained without impact to the team and without significantly changing the way of working.

The entire model created and implemented, intends to implement a control Technical Debt and interest associated with the product, and thus, increasingly improve the quality of software delivered by development teams.

5. Conclusion
It was clear to team members about the importance and benefits of identifying and managing Technical Debt. All participants agreed that Technical Debt identification and management directly affect the quality of the product. Another point to consider is that most members of the teams had never heard about the concept of Technical Debt or just knew intuitively. Now they always think about the necessity of acquiring Technical Debt, and often decide not to do it. They always think and discuss about Technical Debt contraction in the stories to be implemented. And this implies the creation of a culture focused on quality.

Another point well considered, was the simplicity of the process implemented with minor modifications on the existing tool. Technical Debt stories are registered in the tool with other stories, even though this increase the team work to document the stories. Then to start to think about and identify Technical Debt, is not necessary to do many changes in the process or environment of the team.

And the PO of the project have more visibility about the quality of the software, and it becomes clear the importance of to prioritize in some Sprints to pay some debts, for the evolution and health of the product.

According to Eisenberg (2012), Technical Debt management is very important for software developers, especially in agile methods projects. This can be attributed to the changing nature of the code in agile development, the expectation for continuous refactoring, and the need for continuous integration [Eisenberg, 2012] and this research was conducted in a company that uses agile in their development process.

It is also worth mentioning that the main idea of this model is to verify that each iteration is working in the model, and what is not, and then evolve/change whatever it takes.

6. Threats to validity
The research data were collected through interviews and recordings, so, for a future version, would be interesting to perform data triangulation seeking to confront the evidence with data from project documents.

7. Future work
We aim to continue this research, replicating the proposal to other company projects. We also intend to replicate interviews and monitoring the records of Technical Debt in Jira tool to gather more information about the pros and cons of identifying and monitoring Technical Debt.
Additionally, we will suggest and insert new ways and management tools for Technical Debt management and replicate to other teams in the company. Also, other studies with similar objectives are being developed in other companies in São Paulo.

Regarding the Technical Debt, there are still many gaps in how to identify, manage and use it, strategically throughout the project. And, also there is not a complete theory about Technical Debt.

References


