Effects of Technical Debt Awareness: A Classroom Study

Graziela Simone Tonin[1(✉)], Alfredo Goldman[1], Carolyn Seaman[2], and Diogo Pina[1]

[1] Institute of Mathematics, Statistics and Computer Science, University of Sao Paulo, São Paulo, Brazil
{grazzi,gold,diogojp}@ime.usp.br

[2] Department of Information Systems, University of Maryland Baltimore County, Baltimore, USA
cseaman@umbc.edu

Abstract. Technical Debt is a metaphor that has, in recent years, helped developers to think about and to monitor software quality. The metaphor refers to flaws in software (usually caused by shortcuts to save time) that may affect future maintenance and evolution. We conducted an empirical study in an academic environment, with nine teams of graduate and undergraduate students during two offerings of a laboratory course on Extreme Programming (XP Lab). The teams had a comprehensive lecture about several alternative ways to identify and manage Technical Debt. We monitored the teams, performed interviews, did close observations and collected feedback. The results show that the awareness of Technical Debt influences team behavior. Team members report thinking and discussing more about software quality after becoming aware of Technical Debt in their projects.

Keywords: Technical debt · Technical debt awareness · Technical debt impact · Extreme programming

1 Introduction

Several studies have shown that agile methods have provided significant gains in software projects [19]. However, it is also known that when prioritizing delivery speed, as may happen in agile projects, Technical Debt may be incurred. Much of this debt is not even identified, monitored or managed. Technical Debt that is not well managed runs the risk of high maintenance costs.

The term Technical Debt was introduced by Cunningham, who explained it in the following way [4], "…Although the immature code may work fine and be completely acceptable to the customer, excess quantities will make a program unmasterable, leading to extreme specialization of programmers and finally an inflexible product. 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…". Technical Debt is recognized as a critical problem for software companies [2] and has received a lot of attention in the recent years from both practitioners and researchers [16, 17].

© The Author(s) 2017
H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 84–100, 2017.
DOI: 10.1007/978-3-319-57633-6_6

Lim et al. [18] emphasize that: "…most project teams now recognize that Technical Debt is unavoidable and necessary within business realities. So managing Technical Debt involves finding the best compromise for the project team…", but a project team cannot do this if they are not aware of Technical Debt. Also, Lim et al. highlighted that when the development team is not aware of Technical Debt, it will probably result in challenges for maintenance and evolution tasks. Given this scenario, our motivation was to observe the effects of Technical Debt awareness in teams in an academic setting.

The Extreme Programming Laboratory (XP Lab) is a course that has Undergraduate and Graduate students at the University of São Paulo since 2001. The aim of this course is to provide the experience of a real software development scenario using the Extreme Programming values and practices [1].

Extreme Programming emphasizes teamwork; managers, customers and developers are all equal partners in a collaborative team. The main values of Extreme Programming are communication, simplicity, feedback, respect and courage [3].

The objective of our research is to characterize the impact on the team when Technical Debt items are visible, based on team members' perceptions. This study aims to answer the following research question (RQ):

- **What is the impact on the team when Technical Debt is explicitly considered?**

The study was applied in two editions of XP Lab. Four teams were followed in the 2013 edition and five teams in the 2014 edition. We conducted the study and collected data through questionnaires and interviews, and analyzed the source code of the projects with Sonar Qube and Code Climate tools to identify the impact on the teams that explicitly considered Technical Debt (TD).

In the next section, related work is described. In Sect. 3, we describe the context of the Extreme Programming Laboratory. After, in Sect. 4, we provide a description of the research steps, data collection and analysis. Section 5 describes the results. In Sect. 6, we discuss the findings and present the threats to validity. Finally, in Sect. 7, we present the final considerations and future work.

2 Related Work

Few studies deal directly with the technical debt awareness. The study of Kruchten [22] showed that agile teams believe that they are immune to TD, because they use an interactive development process. Therefore, he explains that in these teams, TD items could be contracted rapidly and massively, because code is often developed and delivered very rapidly, without time to devote to good design or to think about longer term issues. This could result in contracting TD items such as a lack of rigor or systematic tests. To deal with TD and to avoid accumulating too much TD, he suggests*: "The first step is awareness: identifying debt and its causes. The next step is to manage this debt explicitly, which involves listing debt-related tasks in a common backlog during release and iteration planning, along with other "things to do."*. Bavani [21] shows that if teams are unaware of the context of meaning of the term TD, they can consider trivial issues or technical tasks as a TD. These teams have to improve the awareness of it, so he proposes

a quadrant to help teams to better recognize and understand the TD concept. The study of Martini [23] listed some causes of architecture technical debt and one of the reasons he found was the lack of awareness about the dependencies between the specific architectural TD and the other parts of the software. Furthermore, there are many related studies on not managing TD and how this affects software quality, such as in the studies of Guo [20], Sterling [24], Li [16], Lim [18], and Curtis [25]. McConnell [26] emphasizes that when a team makes the decision to contract a debt or not, they are really deciding between two ways to complete the current development task, one faster and the other resulting in better quality. Bavani [21] talks about management of TD items in distributed agile teams, and he emphasizes that the management of TD items directly affects the economics of software maintenance and according to him, the key for success in the current global economy is building and maintaining software under optimal costs. Sterling [24] said that TD exists and is detrimental to the maintenance of software quality. Buschmann [27] suggests that teams doing a refactoring in the code should also pay the TD items and improve internal quality. A recent report showed that one of the consequences of incurring TD is the impact on quality [28].

3 Context: Extreme Programming Laboratory

The XP Lab is a regular course offered at the University of São Paulo, to graduate and undergraduate Computer Science students. The motivation is to provide them an opportunity to learn agile software development methods on real projects. In the 2013 offering, there were four teams, with five or six students each. In the 2014 offering, five teams with six students each attended the course. XP Lab students have the support of meta-coaches who are experts in agile methods. They provide agile mentoring for all the teams with the professor's help. Each team also has a coach, who is a student that has more experience in agile methods. The teams develop real projects with on-site customers. The teams have to follow some agile practices, for instance; pair programming, automated tests, continuous improvement, continuous integration, etc. In both studies, the teams worked in pairs and in threes, and the groupings changed many times during the course, sometimes according to the tasks they needed to develop. The course requires a minimum attendance of at least 8 h a week of dedication (four hours in the laboratory and four hours of extra classes), and there is a lunch once a week, to encourage the students' presence in the lab and to allow the students to share experiences. On some weeks, there are short presentations about some difficulties that the teams are facing, where a specialist explains and discusses specific topics. A complete description of the course settings can be found in [1].

3.1 Projects

In Table 1, we briefly describe each of the projects involved in our study:

Table 1. Extreme programming projects

Project	Description
Arquigrafia	Arquigrafia is a public digital collaborative environment, nonprofit, dedicated to the dissemination of architectural images, with particular attention to Brazilian architecture [6].
Games-VidaGeek	A platform for games that support the teaching of programming (with games for Scala, Java, Html, CSS, SQL and other languages being produced) [7].
TikTak	A project focused on collecting feedback data from users and display it in a web dashboard [5].
Mezuro	A framework for monitoring source code metrics [8].
Monitoring system	An online system where students can apply to be a teacher assistant of a regular courses [14].
System specialist in sport	An application to enable researchers working with physiological data to apply metabolic mathematical models [15].
Social networking startups	A social network for Startup, with the goal of creating a community of highly connected and committed entrepreneurs [5].
Family tree	A genealogy community where each individual can create a family tree and from time to time the system attempts to "link" the trees [5].
CoGroo	Portuguese grammar corrector used by LibreOffice [9].

3.2 The Informative Workspace

Each team had its private informative workspace[1] [10, 11], where they physically displayed TD items. In the XP Lab 2013 offering, all teams had a TD board (Figs. 1 and 2). In the XP Lab 2014 offering, each team decided by themselves how to manage the TD items in their informative workspaces. Some teams decided to have the TD board and other teams kept the TD items list on a Kanban board.

3.2.1 Boards

In the TD board (Fig. 1) a team placed the TD items that were incurred and/or identified. On the top of the board, there is a supply of blank cards called 'Fichas'. These cards were used to document the TD items.

Figure 2 shows another team's board where they kept the list of TD items that were incurred and identified. On the right side, they have a reserve of blank cards.

[1] The informative workspace is the place where the teams put all the physical boards and graphics, with the metrics they used to manage the project development also the list of the task they will develop in each sprint.

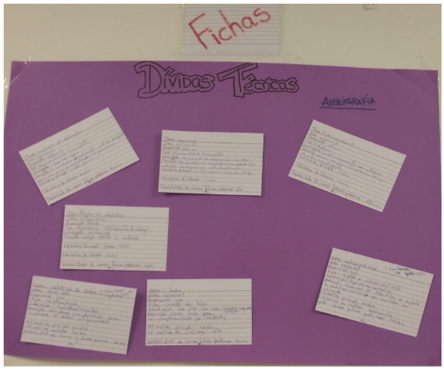

Fig. 1. Technical debt board

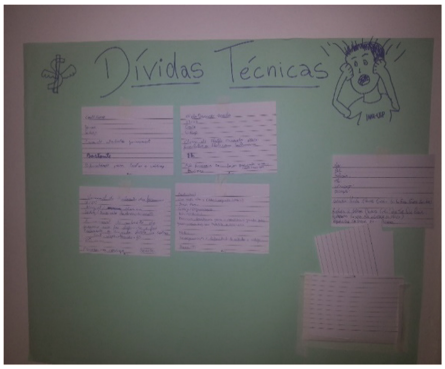

Fig. 2. Board of the technical debt list.

Figure 3 shows one TD item about *duplicated code*. Each card had nine categories to fill out. Below we transcribe the data contained in Fig. 3.

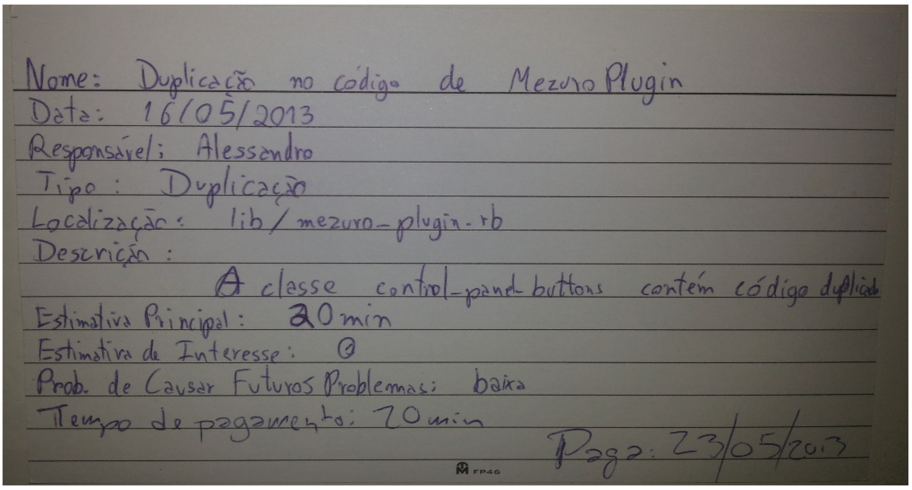

Fig. 3. Technical debt item

In this case the Name of TD item was *Duplicated code in the Mezuro plugin,* the Date (when the item was identified), was 05/16/2013, the Responsible (the person that incurred or found the TD item, in this case Alessandro), the Type was *duplication,* (could be test, documentation, design, etc.), the Location (which part of the code the items was related), was in *lib/mezuro-puglin-rb*. The Description (a brief description of the TD item), *the class control-panel-buttons* has duplicated code. The Estimated Principal, was *twenty minute*s (how much expected time they need to spend now if they implemented that task in the correct way, if they did not know how much time, then they could use a scale of *high* - if they probably will spend a large amount of time -, *medium* - if they probably will not spend much time - and *low* - if they probably will solve it quickly).

The Estimated Interest Amount should be filled out when they pay the TD item. The Probability of being a future problem (i.e. the interest probability) in this case was *low*. In this case, they also added in the card the Date when they paid the TD item, *05/23/2013* and how long it took them to pay off the item, also *twenty minutes*.

These boards represent some of the boards used in the team's informative workspace. Some teams used a specific board to manage TD, as shown in Figs. 1 and 2, other teams used the Kanban board and put the TD items together with the tasks of the sprint. Furthermore, some teams also placed the list of TD items in the tool used to manage the project.

3.2.2 Tools

The teams had two code quality analysis tools:

- Code Climate: a tool for quality analysis of code repositories (https://codeclimate.com/).
- Sonar Qube: a code quality analysis platform that has a plugin that identifies TD (http://www.sonarqube.org/).

4 Research Methods

Data was collected and analyzed for this study through interviews and questionnaires. Before data collection, the teams spent some time identifying TD items in their projects. Below, we first describe how TD items were identified, and then we describe our data collection and analysis methods.

4.1 Technical Debt Identification

In the two offerings of the XP Lab, we followed slightly different steps to help the teams to identify TD.

4.1.1 XP Lab 2013

Four weeks after the students started working on their projects, we gave a presentation about TD, as some students were familiar with the term, but others were not. After the presentation, we had a discussion and we encouraged the students to talk about their views of TD. We also talked about some concrete examples they had in their projects. The discussion lasted for about one hour and after that, each team had to prepare a TD board, where they would have to document their TD items (Fig. 3 shows an example of a TD item). Each TD item was written on a card, and in this card they had to fill out a list of topics, after that it was then pinned to the board. The card structure was based on the template developed in [12]. Each team had a board with a set of cards representing TD items. Every team that identified or incurred a debt put the information on the board.

4.1.2 XP Lab 2014

In the XP Lab 2014 offering, we made a presentation about TD for all the students together for 30 min, about two months after the course began. After that, we discussed it for another 20 min, during this time the students could clarify their doubts about TD. The students already had the code quality analysis tools available, Sonar Qube and Code Climate, since the beginning of the course. We did not impose the use of either the boards or the tools to identify TD. We showed them the meanings of TD items and some examples in each project. We also presented examples of a TD board and of the categories they could use to identify TD items. Then each team decided whether they would monitor TD on their project or not.

4.2 Second Step – Interviews and Questionnaires

Data collection was done differently in the two XP Lab offerings. In both cases, similar data was collected both at the beginning and at the end of the course.

4.2.1 XP Lab 2013

Eight weeks after the teams started to identify TD items and fill the boards with cards, we carried out a face-to-face interview with the pairs in each team. The interview motivation was to verify the influence on the team of the TD visibility. The interview was composed of twenty questions, with open-ended and multiple-choice questions, separated into the following topics[2]:

- The concept of TD.
- Were there any changes in the software development process?
- Negotiation with clients.
- About the experience of identifying TD.
- What is the relevance of identifying TD?
- What is the impact on software quality?
- Do the teams pay off TD?
- Will the teams pay off some TD?

Four weeks after the first interview, at the end of the course, we did the last interview with an open format and we performed it for each team. In the last interview, each team was invited to talk about the experience of making TD explicit. Each interview took about twenty minutes.

4.2.2 XP Lab 2014

In this edition of the course, we decided to apply a questionnaire on what each team member thinks about TD (the questionnaire was answered by the students individually; this approach was taken to try decrease a possible bias). This questionnaire was applied

[2] It is possible to access all the questions in the following link https://www.dropbox.com/sh/gen3dr97xxofs21/AACo11oqbBsaCprOCQtSYv5Ja?dl=0.

one week before the class received a talk about TD. We sent a link to the questionnaire by email and then the students had one week to answer it.

The questionnaire was composed of seventeen open-ended and multiple-choice questions, separated into the following topics[3]:

- Software quality
  - What does the team do about quality?
- Familiarity with TD.
  - Do you know about and use the TD concept?
- How TD is used in the project.
  - Are you using the Sonar Qube or Code Climate report?
  - Are you using a TD list?
  - Have you paid any TD item?
  - Is there any evidence that having the TD items visible has an influence on the team?
- Did you identify any TD item that was not identified by the tools?
- Are you going to consider TD in future projects?
- Do you think the TD concept is relevant?

The same questionnaire was applied a second time at the end of the course. The aim was to see if there was any change in the team members' behavior.

4.3 Third Step – Data Analysis

For the data analysis, we used coding techniques from the grounded theory approach [13]. Grounded theory methods are aimed at building or discovering a theory. In this approach, the data analysis proceeds in three interdependent steps: open coding, axial coding, and selective coding. In the open coding step, the researcher interprets the data to identify patterns and define codes, "…event/action/interaction, are compared against others for similarities and differences; they are also conceptually labeled […] conceptually similar ones are grouped together to form categories and their subcategories…" [13]. In axial coding "…categories are related to their subcategories, and these relationships tested against the data…" [13]. Then in selective coding "…all categories are unified around a central 'core' category and categories that need further explanation are filled-in with descriptive details…" [13].

For data analysis, we used the NVivo[4] tool, which is widely used for analysis of qualitative data. In this case, the goal was not to use grounded theory to develop a new theory but only use its coding steps to answer our research questions.

We also analyzed the source code of the projects with the Sonar Qube and Code Climate tools, to try to identify relationships between team's beliefs and the reports from these tools.

[3] It is possible to access all the questions in the following link https://www.dropbox.com/sh/gen3dr97xxofs21/AACo11oqbBsaCprOCQtSYv5Ja?dl=0.

[4] http://www.qsrinternational.com/support/downloads/nvivo-9.

5 Results

In this section, we describe our findings organized by the coding steps. As the main question in the both editions of XP Lab was the same and the obtained results were similar, we analyzed the results of both editions together.

5.1 Open Coding

In this step, the data analysis was conducted by reading the transcripts of the interviews and also the answers from the questionnaires. We applied the coding process to this material, line by line. In this phase, we discovered the open codes. In Table 2, we list three code samples. It is possible to access the list of the open codes that emerged from this first codification in an appendix[5].

Table 2. Example of codes resulting from open coding

Open codes	What they talked about
Changed attitude of the teams	The team discussed more the tasks they have to do before incurring a TD and they thought more before taking the decisions.
Communication	After the identification of TD items, the team had more discussions.
Maintainability	The identification of TD items helps the teams to know that there will be some changes in the software in the future.

5.2 Axial Coding

The open codes were reassembled in new ways during axial coding to form categories. The goal was to create a higher abstraction level. Thus, codes were grouped to form subcategories, and in turn, they were organized into categories. This process was highly iterative, with codes and categories forming and re-forming as more data were incorporated into the evolving understanding [13].

In Fig. 4, it is possible to observe the list of categories and subcategories resulting from axial coding analysis. The first level is the main category resulting, this category emerged from the subcategories of the second level, the subcategories are resulting from the codes emerged in the third and fourth level.

One of the most important influences when we make a list of TD items is the *attitude of the team (team behavior)*, "…registers by not forgetting, there was a change in the attitude of the team…" and "…increased people's concern regarding the Technical Debt…". The team had less 'untouchable' expert professionals and behaved more as a whole team. They *talked more about the TDs* "…we discussed these debts. Otherwise, the project would not have advanced…" and thought about the necessity of incurring it. *It helped* them to have the same understanding of the concept of TD because they discuss it (*TD concept*). In addition, if the team members were not sure whether to incur

[5] It is possible to access all the codes in the following link https://www.dropbox.com/sh/gen3dr97xxofs21/AACo11oqbBsaCprOCQtSYv5Ja?dl=0.

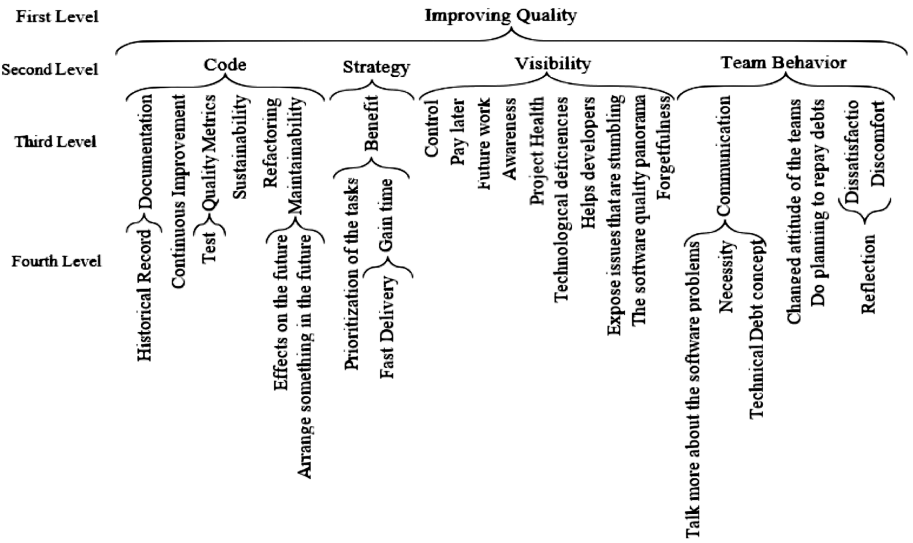

Fig. 4. Categories and subcategories of the XP Lab 2013 and 2014 edition

a TD item or not, the team member debated with another team member to help him to take this decision, thus improving the team's *communication*. Team members started *talking more* with each other and because of it, they knew what part of the project was being modified and the problems of the software, "…It was easier to remember that we have to fix things, debts…". Furthermore, if team *communication* was good, they were more comfortable to share with each other their difficulties. After the team began to identify TD, developers discussed their decisions rather than just doing something and moving on, now that all the software problems were more clear to the team members, "…usually only the person thought or knew about it (…), now with the TD it becomes clearer as well…". They began to argue among themselves, before incurring a debt, "… As evidenced here we even got everyone talking about the debts, instead of just looking to give a quick solution and move along…".

They started to *think more about* if it was *necessary* or not to incur some TDs. Several times they concluded that it was not required, a team member says: "[…] From the moment I started to think about that item, which was debt, I asked myself about what is the current cost, compared to the future cost. Because if the cost of doing now is less, then it's better to fix it now…", other student says: "…we think twice before making a TD…".

When TD items were visible, the team had more *control*, over whether they will pay off the TD item, whether they will incur more debts or not, or whether they will incur and *pay later*. Moreover, this facilitates planning to repay the TD items, "**…** we analyzed some of the debts, now we will plan, what we might kill, to kill some of those debts, then it becomes easier to make this analysis…". Therefore, if they incur a TD item, they would make it visible, would monitor it, and sometimes they would look back at this TD item. This way, they always thought about *continuous improvement*. Thus, the team could be defining a strategy to pay off or not some TD items to improve the *code* quality.

However, if the team member incurred a TD item, and never paid, the project would probably lose *quality.* Nevertheless, they might incur TD to prioritize other tasks or as a business decision to *deliver fast* and then used it *strategically*.

The TD item list presented indications on whether the code quality was improving or not, and helped them to understand the current development state It also indicated if they would have a lot of future work and future *refactoring* to do in that code. Indeed, if the software had TD items, the team would probably need to perform some *refactoring*, and it directly affected the *sustainability* of the project. Furthermore, the team could do an analysis of the TD items and they could be defining a software's *quality metrics* to help them monitor the software quality, for instance, *test*. They also could identify *technological deficiencies* and define possible directions to improve their technical skills and not repeat the same mistakes, in order to mitigate the occurrence of those TD items that were recurrent.

The teams used the TD item list as *documentation* providing a *historical record* of the immature parts of the project. Therefore, each team knew that some parts of the project should be improved; it was possible to see if there was a TD item to pay off. This *documentation* helped the teams to maintain the code and it reflected on the *health of the project.* In addition, it affected the teams by sometimes causing *dissatisfaction* with the *quality* of the software. Many team members saw that it was very *uncomfortable* to arrive at work and see that the *health of the project* was not so good. If the source code health of other teams was good, it was even more *uncomfortable*. Therefore, this process of the team having discussions, and having *dissatisfaction* impacted in the following *reflection:* when a team member decided to incur or not TD, thought and discussed it, he better understood the problem that he had to solve. This often resulted in the non-insertion of a TD item in the code, since it only lacked the understanding of what should be done.

Considering TD implied generating a culture focused on *quality*. It affected factors related to project continuity. It has an influence on the cost and the viability of *maintainability* and evolution of the project. A developer said that, if they did not have the TD items *visible* it was so difficult to identify *the software quality landscape,* "…it was difficult for you to identify the whole landscape…". Therefore, if in the *future* the team needed to make some changes in the legacy code, they already knew what they were and where the problems were located. It provided a general *awareness* to the team about the problems of the software. It also might help a new team member that did not know about the code to have a notion on the quality of the code and the location of the code problems.

5.3 Selective Coding

Selective coding constituted the third stage of data analysis, with the objective to refine and integrate categories, unveiling a category deemed as central, encompassing all the others. The full potential of abstraction was employed to incorporate the full scope of the data investigated and coded [13].

In our case, the objective was not to generate a theory, but rather to identify the main categories. The aim was to describe the impact of TD awareness.

The categories, *Strategy, Team Behavior, Code*, and *Visibility* represent the main influences on teams due to making TD items explicit. Each of these categories captures part of our results, although none of them describes the phenomenon entirely. For this reason, another abstract category is required, a conceptual idea on which all categories are included. As such, we concluded that the resulting core category might be a perceived notion on ''*Improving Quality*".

All teams progressed towards creating a culture of *Quality* of the *code*, *team,* and *project*. This arose primarily because each team member started to think more about the need to incur TD items. Many times they decided that incurring TD was not necessary in a given situation. When a team member was not sure about the necessity or not to incur the debt, they spoke with other members to make a decision. Therefore, the team improved their communication and then it was clearer what each member was doing. So, it was easier to understand the objective of the project. Then, making TD explicit has a direct implication on the *Team Behavior*. Most of the team members said that after they started to identify TD, they talked more with each other, thought more about the real need to incur debt or not, discussed more about code quality, refactored more frequently some parts of the code, knew where the code problems were, and where each team member was working at any time on the project. In addition, when communication among the team was good, people felt more comfortable to expose and discuss their problems with the team. The team then became more a group that works together, rather than a group of experts on different parts of the system.

In some situations incurring some TD items was a *Strategy* to gain some time, due to the time to market. Moreover, if there was a list of TD items it might be possible in the future to correct them, by refactoring. The list of TD items was used as documentation, this enabled the historical record of the TD items list that the code contains. When the need to change a particular part of the system appeared, it was possible to verify if it had some TD item and if this debt would affect such functionality. Therefore, the team members had the option of paying it off or not. The documentation helped in the *Visibility* of the project's health. If the software had any TD, probably it would have more deficiencies.

Finally, when a team had the TD items list they automatically became *Aware* of the TDs of the project, consequently about the software quality. So, the team could think about it, before possibly incurring in another TD item. The team could decide when and where they would improve the software quality.

5.4 The Code Analysis with the Tools

In XP Lab 2013 edition, we analyzed one project with Code Climate tool (supported the language used in the project, in this case, Ruby). Further, we analyzed two projects with the Sonar Qube tool. In the XP Lab 2014 edition, we also analyzed two projects with Code Climate tool and four projects with the Sonar Qube. We considered the Code Climate metric, grade point average[6] (GPA) and in the Sonar Qube the following metrics: code smells, security, reliability, maintainability, duplications, documentation,

[6] https://docs.codeclimate.com/docs/code-climate-glossary#gpa.

issues, technical debt rating, complexity, size, duplicated blocks, bugs & vulnerability and duplications. The GPA of the Mezuro's code and the Monitoria project increase after the teams considered TD. Then the quality of project increased, it means that the remediation (the amount of effort required to improve a software issue), was 0 to 2 M (too short). In the analysis with Sonar Qube of the projects: Game VidaGeek, Tiktak, Arquigrafia, Family Tree, Social Networking Startups and Specialist in Sport we did not identify large variations in the measured metrics comparing the two different versions of each project (Before and after they consider TD). However, in 5 of the 6 projects, the rate of duplication and the duplicated blocks decreased after the team started to consider TD, indicating an improvement in code quality.

It is important to highlight that the students said that most of TD items were not identified by the tools, which explains the modest size of the changes in these metrics. For instance, one team was using Handsontable[7] for data entry, and they had a problem with the validation of the data. The presence of this type of TD item, nor the impact of paying it off, could not be identified with Sonar Qube or Code Climate reports. In general, many types of TD items (e.g. those related business rules) cannot be identified through static metrics.

6 Discussion of the Findings

The teams had some similar views on the importance and benefits of making TD explicit. A significant finding is that the teams considered it very helpful because they could see the whole landscape of the software quality (they knew which part of the software had immature code). They also emphasized that it was very useful to have a board where every day they could see the health of the code. Before becoming aware of TD, the team members reported that they sometimes incurred TD but never remembered to go back and correct it. But after considering TD, they thought about the necessity of incurring TD and often decided against it. Also, they could see the TD list and so they did not forget the TD items that needed to be addressed. They discussed more about how to implement the tasks, also they talked more about the problems of the software because they had the list of the TDs visible. This process of thinking about incurring or not TD, discussing about it and reviewing the TD during the project can create a culture focused on improving the software quality.

In addition, in this study we explored some ways of identifying and monitoring TD. Our subjects found some form of a TD board very useful for documenting TD, making it visible, and adjusting both the TD board and their behavior accordingly. By using the TD board, they always know the list of software deficiencies so have a constant reminder of how to organize their work and improve the software. As a complementary aid, they may use tools to help them to identify and monitor TD occurrence. However, it is important to highlight that tool reports provide a static analysis of the software quality and some TD item could not be identified using static metrics.

[7] https://handsontable.com/.

The results of this study could motivate teams to consider TD further, to help developers convince leaders and directors, the decision makers, to start considering TD. These approaches used by the XP Lab teams, such as boards, cards and tools can help teams in companies to deal with TD. In addition, they could define the list of TD items that are crucial to the project but hard to identify with the tools. As a result they can define a strategy to deal with the TD over time.

6.1 Threats to the Validity

In this study, we took some actions to mitigate possible biases, we describe these in the following points:

- *Construct validity (credibility):* We used multiple data collection approaches with the aim to reduce possible bias. When planning the interviews and the questionnaire we discussed the best way to formulate the questions. We did a first interview and questionnaire with one member of the group as a pilot test. Based on this test we reviewed the questions. We did not include these data in the final analysis. One thing that it is important to highlight is that the students might not have understood the main meaning of the questions correctly, in the interviews and questionnaires. Because of this, in the interviews, if the student did not understand the question the interviewer explained the question for them. The researcher was available throughout both studies if the students had any questions.
- *External Validity:* This study can be replicated in other academic courses, also in companies. In both cases, the study can be separated into two parts and can be applied in these situations: one in teams that do not consider technical debt yet, to verify if awareness of TD influences something in the team, such as communication. Furthermore, this study can be analyzed with teams that already consider technical debt, by identifying, monitoring and managing if it is possible to identify some changes in the team behavior and in the software quality. If they have a historical record, we could also measure the software quality with tools. Finally, to carry out this study is not necessary to make significant changes in the team's environment, which makes feasible to replicate in companies.
- *Internal Validity:* We analyzed the data separately, first the interview transcriptions, then the questionnaire responses, and then compared and merged the findings that were relevant and had a lot of evidence in the results of both studies. In the case of any doubt about a specific point, we went back to the data and re-analyzed them. After that, the advisor and co-advisor read the results and if they indicated some points to be re-analyzed, the researcher re-analyzed the data. We did analysis and re-analysis many times until we were sure of the conclusions.
- *Reliability:* To interpret the data we followed the coding techniques from the grounded theory steps. Also, the data analysis was made by a single researcher, however, the results of the analysis were discussed by the two researchers and with the advisor and co-advisor, every time a doubt arose the data were re-analyzed. Also, this paper is a result of an analysis of the data that lasted two years, where the researcher compared the data many times. Furthermore, the preliminary results were

presented and discussed at seminar[8] attended by top researchers in this field. It is important to observe that when we infer that awareness of TD could impact software quality, we are describing the perceived quality by the team.

- *Objectivity:* The results show the information derived from the data, the codes and categories emerged were related with the data quotes.

7 Final Considerations and Future Works

This work describes results about the influences of making TD explicit in an academic setting. Our results show the importance of making TD visible and how that influences teams. It is important to point out that no negative influences were identified. The team members were always very excited about the results of making TD items visible. As communication in the team was improved, all team members thought more about quality, not just specific members. The "agile" culture of the teams improved and in addition, the team believed that it was easier to show the impact of the TD level to clients, showing that it is possible to invest some time to improve the quality. The main results emphasize the Extreme Programming values and helped the teams to support values such as communication at all levels, courage to change and feedback to continuously improve the software.

In future work it is important to verify the influence on the team in the long term, especially concerning speed and code quality. It is also important to create ways to compare the perceptions of the developers with the results of the tool reports. For instance, in these studies the students believed that when they started considering TD, the project quality improved, but when we analyzed the code with the tools, we saw that the reports did not indicate significant changes in source code quality. Then, it is interesting to investigate why this happened, and possible future solutions.

Acknowledgments. We are grateful to all the students and TAs of the XP Lab course (2013 and 2014 offerings) for providing valuable data for this research. We would like to thank IBM, CNPQ, CAPES and FAPESP, too, for funding this work.

References

1. Santos, V. et al.: Uncovering steady advances for an extreme programming course. CLEI Electron. J. **15**(1) (2012). paper 1
2. Edith, T., Aybuke, A., Richard, V.: An exploration of technical debt. J. Syst. Softw. **86**, 1498–1516 (2013)
3. Kent, B.: Extreme Programming Explained: Embrace Change. Person Education Inc, United States (2005)
4. Ward, C.: The WyCash portfolio management system. In: Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications, pp. 29–30 (1992)
5. Extreme Programming Projects – CCSL. (http://www.ccsl.org.br/oldwiki/index.php)
6. Arquigrafia. http://www.arquigrafia.org.br/. Accessed May 2016

[8] http://www.dagstuhl.de/de/programm/kalender/semhp/?semnr=16162.

7. VidaGeek. http://aprenda.vidageek.net/. Accessed May 2016
8. Mezuro. http://mezuro.org/pt. Accessed May 2016
9. CoGroo. http://ccsl.ime.usp.br/cogroo/. Accessed May 2016
10. Oliveira, R.M., Goldman, A., Mello, C.: Designing and managing agile informative workspaces: discovering and exploring patterns. In: Proceedings of the 46th Hawaii International Conference on System Sciences (2013)
11. Oliveira, R., Goldman, A.: How to build an Informative workspace? an experience using data collection and feedback. In: Agile Conference (2011)
12. Seaman, C.: Technical Deb Minicourse. At the University of São Paulo (2013)
13. Corbin, J., Strauss, A.: Grounded theory research: procedures, canons and evaluative criteria. Zeitschrift fur Soziologie **19**(6), 418–427 (1990)
14. Monitoria. www.monitoria.ime.usp.br. Accessed May 2016
15. System Specialist in Sport. http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0145733. Accessed May 2016
16. Li, Z.G., Avgeriou, P., Liang, P.: A systematic mapping study on technical debt and its management. J. Syst. Softw. **101**, 193–220 (2014)
17. Poliakov, D.: A systematic mapping study on technical debt definition. Lappeenranta University of Technology School of Industrial Engineering and Management Degree Program in Computer Science (2015)
18. Lim, E., Taksande, N., Seaman, C.: A balancing act: what software practitioners have to say about technical debt. IEEE Comput. Soc. Softw. **29**(6), 22–27 (2012)
19. VersionOne.: State of Agile Report (2015). http://info.versionone.com/state-of-agile-report-thank-you.html
20. Guo, Y., Seaman, C.: A portfolio approach to technical debt management. In: Proceeding of the 2nd Workshop on Managing Technical Debt (2011)
21. Bavani, R.: Distributed agile, agile testing, and technical debt. IEEE Softw. **29**, 28–33 (2012)
22. Kruchten, P., Nord, R.L., Ozkaya, I.: Technical debt: from metaphor to theory and practice. IEEE Softw. **29**, 18–21 (2012)
23. Martini, A., Bosch, J., Chaudron, M.: Architecture technical debt: understanding causes and a qualitative model. In: Proceedings of the 40th Euromicro Conference on Software Engineering and Advanced Applications (2014)
24. Sterling, C.: Managing Software Debt: Building for Inevitable Change. Addison-Wesley Professional, Boston (2010)
25. Curtis, B., Sappidi, J., Szynkarsky, A.: Estimating the size, cost, and types of technical debt. In: Proceedings of the IEEE 3rd International Workshop on Managing Technical Debt (MTD 2012) (2012)
26. McConnel, S.: Managing Technical Debt. http://www.construx.com/File.ashx?cid=2797. Accessed April 2008
27. Buschmann, F.: To pay or not to pay technical debt. IEEE Softw. **28**(6), 29–31 (2011)
28. Avgeriou, P., Kruchten, P., Ozkaya, I., Seaman, C.: Managing technical debt in software engineering. Dagstuhl Rep. **6**, 110–138 (2016)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.