Managing Technical Debt: An Industrial Case Study

Zadia Codabux, Byron Williams
Dept. of Computer Science and Engineering
Mississippi State University
Starkville, MS, USA
zc130@msstate.edu, williams@cse.msstate.edu

***Abstract*—Technical debt is the consequence of trade-offs made during software development to ensure speedy releases. The research community lacks rigorously evaluated guidelines to help practitioners characterize, manage and prioritize debt. This paper describes a study conducted with an industrial partner during their implementation of Agile development practices for a large software development division within the company. The report contains our initial findings based on ethnographic observations and semi-structured interviews. The goal is to identify the best practices regarding managing technical debt so that the researchers and the practitioners can further evaluate these practices to extend their knowledge of the technical debt metaphor. We determined that the developers considered their own taxonomy of technical debt based on the type of work they were assigned and their personal understanding of the term. Despite management's high-level categories, the developers mostly considered design debt, testing debt and defect debt. In addition to developers having their own taxonomy, assigning dedicated teams for technical debt reduction and allowing other teams about 20% of time per sprint for debt reduction are good initiatives towards lowering technical debt. While technical debt has become a well-regarded concept in the Agile community, further empirical evaluation is needed to assess how to properly apply the concept for various development organizations.**

***Index Terms*—technical debt, Agile methods, industrial case study, Scrum, semi-structure interviews**

I. Introduction

Software development is prone to failure. One of the root causes of this failure is the use of sequential design processes for building complex software intensive systems. Sequential processes work when the requirements are defined upfront and when the remaining software development activities are instituted based on the initial requirements (e.g., design, implementation, testing). Traditional software development processes, such as the waterfall model, are not the most appropriate when business needs and technology change rapidly. Many development groups often deviate from this normative waterfall process. Their focus would shift to the product and the customer needs rather than the plan. This shift gave rise to Agile software development. Williams and Cockburn [13] state that Agile development is "about feedback and change," and that Agile methodologies are developed to "embrace, rather than reject, higher rates of change".

One of the primary benefits of Agile software development is the quick release of software functionality. The focus on functionality often lessens the focus on design, good programming practice, test coverage, etc. By focusing on functionality, an obligation arises for the developer to go back and complete these items neglected for the sake of functionality. This phenomenon is known as technical debt.

While the concept of technical debt has been existent for some time, it has been the adoption of Agile development methods that has given the term its visibility. Agile methods started to grow in popularity in 2001 following the signing of the Agile Manifesto. Since then, technical debt has become an increasingly important concept in software engineering research. Seaman et al. gave several examples of the consequences of not "paying off" technical debt such as large cost overruns, quality issues, inability to add new features without disrupting existing ones, and the premature loss of a system (i.e., the software becoming unusable before its expected lifetime is over) [9].

Time constraints often prohibit all software development tasks from being accomplished; therefore, the tasks must be prioritized. The technical debt must also be accounted for in deciding upon priorities. It is important to understand that it is not always a bad practice to take on debt. Technical debt is acceptable when the priority is getting new functionality up and running, but it is managing the debt that has presented problems to the practitioner community.

The overall goal of our research is to study Agile development, particularly technical debt in an industrial context. The objective is to determine the best practices and known challenges with Agile adoption and the management of technical debt. To achieve this, we conducted 3 on-site process evaluations with our industrial partner since their initial adoption of the Agile methods. For this paper, our focus is on technical debt. Despite the increasing interest in the technical debt metaphor, procedural knowledge about the topic is mostly described in blogs and there is a limited amount of quality research to evaluate claims and further understanding. This study highlights the insights of software practitioners on technical debt. The aim is to contribute to the understanding of the technical debt metaphor and its use in software development organizations.

Section II highlights background and related work. Section III focuses on the research methodology adopted for this work. Section IV presents the results. Section V provides insights on the results and discusses the limitations of this study. Section VI concludes and outlines the future work.

II. Related Work

This section presents related research and documented best practices from practitioner sources on technical debt management. Particularly, we report on research and relevant posts identifying the types of debt, addressing the decision to accept debt, estimating the financial impact of debt, and prioritizing technical debts. In addition, two empirical studies evaluating technical debt are presented.

Generally, before determining whether to accept technical debt, one must first gather appropriate information. Doing so requires the practitioner to know the different types of technical debt that can be accumulated.

A. Technical Debt Taxonomy

Several attempts to define a taxonomy of technical debt have been made. McConnell classified technical debt as intentional and unintentional. Unintentional debt is described as debt incurred inadvertently due to low quality work (e.g., a junior programmer writing bad code that does not conform to recognized coding standards). Intentional debt is described as debt incurred deliberately such as postponing proper reconciliation of databases by writing some glue code to synchronize them. Another example of intentional debt is the planned release of software that does not match the coding standards with the expressed notion that it will be cleaned up later. The intentional debt category can be further viewed as either short-term or long-term debt [6].

Fowler defines a technical debt quadrant which consists of 2 dimensions: reckless/prudent and deliberate/inadvertent (see Fig. 1) [4]. The types of technical debt as indicated by Fowler are as follows:

- Reckless and deliberate debt
- Prudent and deliberate debt
- Reckless and inadvertent debt
- Prudent and inadvertent debt

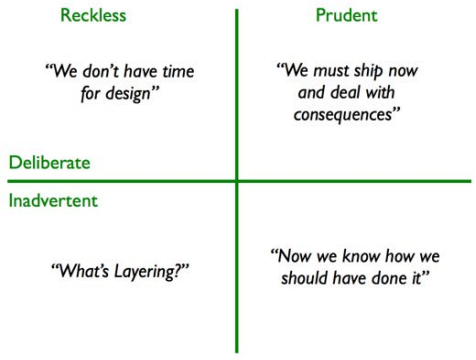

Fig. 1. Technical Debt Quadrant [4]

One can also classify technical debt based on its association with traditional lifecycle phases. Documentation debt, design debt, coding debt, testing debt, and defect debt are not exclusive to, but all are related to the phases of a traditional waterfall lifecycle. Documentation debt is incurred when documentation is not up-to-date or adequate. Design debt–which has also been categorized with code debt –is debt that is associated with the source code (e.g., modularity violations, code smells, and grime). Testing debt is incurred when tests are not implemented or executed despite being planned. Defect debt is incurred when known defects are not fixed [10]. These taxonomies require more thorough investigation into how they are used in practice and have not been standardized.

B. Technical Debt Cost Estimation

The cost of technical debt has been modeled after the concepts of principal and interest borrowed from the finance discipline. Principal, in this case, is the cost of eliminating the debt (i.e., performing the technical work necessary to fix) at the present moment. Interest is the additional cost of not eliminating the debt now; therefore, the interest will be paid when the debt is addressed.

Ward Cunningham also describes the technical debt metaphor in financial terms, introducing the concept of interest probability as the probability that the debt will make other tasks more expensive over time if it is not paid [1].

Nugroho et al. defines technical debt as the cost to improve software quality to an ideal level. Interest is defined as the additional cost of software maintenance for not achieving that ideal quality level. He calculated debt based on Repair Effort (RE), or the repair cost to achieve ideal quality. He further surmised that RE is dependent upon Rework Fraction (RF), Rebuild Value (RV) and Refactoring Adjustment (RA). The interest is estimated as the Maintenance Effort (ME), which is dependent on the Maintenance Factor (MF), Quality Factor (QF) and RV [8].

S. Chin et al. break down the cost of technical debt into Principal, Recurring Interest (RI) and Compounding Interest (CI). As mentioned above, principal is the cost of completing right now. RI, which is more specific to the organization, is the cost incurred by holding onto the debt. RI considers factors such as the cost of resolving customer defects. CI is defined as the additional technical debt that accumulates over time due to non-repayment [12].

Curtis et al. presented an approach for calculating technical debt's principal by assessing coding violations with regards to robustness, performance efficiency, security, transferability and changeability. The code violations are classified as low, medium, and high severity violations. The principal is calculated as a function of the number of should-fix violations in the software, the hours to fix each violation, and the cost of labor [2].

There is no generally accepted scheme or standard for calculating principal, and interest as this research is still in its early stages.

C. Decision Making using Technical Debt

The different options concerning whether to pay off debt or perform enhancements and other maintenance activities to correct bugs are presented below.

Seaman et al. proposed 4 distinct decision–making approaches for prioritizing technical debt. In the Cost Benefit Analysis approach, the principal, interest, and interest probability of each technical debt item is assigned ordinal

scales of measurements such as low, medium and high. This approach supports coarse-grained preliminary decisions on debt action. For example, a company may decide to address 75% of their debt with high interest, 25% of their debt with medium interest and defer the ones with low interest. The Analytic Hierarchy Process (AHP) assigns weights and scales to different criteria that are used to measure technical debt. Then a series of pair-wise comparisons are performed between the alternatives to get a prioritized ranking of the technical debt items. Based on this technique, the items on top of the list should be dealt with first. The Portfolio approach targets maximization of the return of investment value and investment risk minimization to decide the order in which technical debt is addressed. The Options approach is analogous to investing in refactoring the debt item with the long-term objective of facilitating maintenance in the future, thereby saving money [9].

Snipes et al. describe the decision to fix or defer debt as dependent upon the amount of technical debt that has been accumulated as defects. They identified factors that influence the decision, which include:

- Severity
- Existence of a workaround
- Urgency of the fix required by a customer
- Effort to implement the fix
- Risk of the proposed fix
- Scope of testing required.

The factors are listed in decreasing order of importance with severity being the most influential factor [11].

The above techniques should be empirically evaluated to assess their validity and to gain better insights on how they can be improved. Further examination to determine how practitioners consider decision-making is also warranted. This study attempts to assess decision-making from a practitioner's standpoint.

We have found only two empirical studies on technical debt carried out in an industrial context [3][5]. Klinger et al. describe an industrial case study carried out at IBM. The goal was to find out how technical debt is analogous to financial leverage (the extent to which borrowed money is being used in a business or investment). The goal is to give the organization the flexibility to incur debt intentionally and to pursue options that would not be possible otherwise. The researchers interviewed four technical architects to examine how the decisions to incur debt were taken and the extent to which the debt provided leverage. The findings of the study pointed out the organization's failure to assess the impact of intentionally incurring debt on individual projects. Decisions regarding technical debt were rarely quantified, and organizational gaps among the business, operational, and technical stakeholders contribute to incurring debts [5].

Lim et al. interviewed 35 software practitioners to obtain their insights on how they characterize, perceive and understand technical debt as well as the context in which it occurs. The study showed debt intentionally being incurred for some short-term benefits such as meeting budgets and delivery constraints while accumulating long-term problems such as decreased software quality. The findings recommend increased communication about technical debt and making it more visible to all the stakeholders [3].

Our study adds to this technical debt literature through our observation of a larger pool of developers (~250) during their end and start of PSI, retrospective meetings and sprint planning. In addition to defining and characterizing technical debt, we are also examining how technical debt is incurred and what current practices are used for prioritizing debt.

III. Methodology

This section describes our research methodology. We present the research goal, the questions, and the data collection methods, and we describe the environment in which the study was conducted.

Our overall goal was to conduct an industrial case study to gain software development team members' insights on Agile software adoption and how technical debt affects the development process. This study reports the initial findings of the research conducted in 3 separate 3-day visits to our industrial partner from April-November 2012.

The following questions were posed upon commencement of our research. A brief description of our rationale is provided after each question:

RQ1: How can technical debt be characterized to distinguish the impacts of certain types of debt?

The rationale for this question lies in understanding the distinction between the different types of debt. Our goal is to determine how to handle various types of debts based on their anticipated consequences. A technical debt taxonomy will assist practitioners to understand how to evaluate different types of debt.

RQ2: What are the consequences of technical debt on the development process?

We want to understand the impact of long-term and short-term debt management decisions. Do managers consider the more immediate implications of debt (e.g., subsequent sprints/iterations) and longer term implications (e.g., releases, new software versions)?

RQ3: How is technical debt addressed?

We are interested in understanding the different techniques and strategies practitioners use to reduce technical debt.

RQ4: How can technical debt be prioritized so that the most critical ones are addressed first?

Based on the results of the previous questions, RQ4 focuses on determining which types of debt are more critical than others. Answers should inform a prioritization scheme to help decide which debts should be addressed first. Practitioners can then better allocate limited resources towards reducing technical debt.

Our study was carried out with an industrial partner, a leading global provider of networking and communications equipment. One software development division of the organization primarily focused on adopting Agile methods, particularly Scrum and established 28 scrum teams. The Agile process was instituted approximately 20 weeks prior to our

initial visit. Prior to then, only 2 teams within the division had any formal experience with Agile at the organization.

A. Study Design

We used participant observation, semi-structured interviews, and a questionnaire to gather data for this study. The study consisted of 3 visits to the partner facility within a 10-month period. Our participant observations took the "fly-on-the-wall" approach to observing meetings related to Agile implementation including sprint planning, review, retrospective, and the larger Scrum of Scrum meetings within the entire division. The semi-structured interviews were used to assess the practitioner's (both management and technical) viewpoints of technical debt and to understand the terminology used. The interviews were of both individual developers and Scrum teams (6-9 team members). We used a questionnaire to obtain background information (e.g., work experience) on each interview participant. Additional details are elaborated below.

B. Data Collection

This first phase of the study was carried out over a 3-day period to investigate the adoption of Agile methods at the organization. We met with members of the Agile adoption team to assess their approach to incorporating Agile throughout the 250+ member development division under review (of about 1800 employees). The group adopted Scrum, and the 3-day visit included the Scrum of Scrums meeting that involved all Scrum teams. Our visit coincided with their 2nd 10-week Potentially Shippable Increment (PSI) meeting. The PSI meeting takes place every 10 weeks and includes a 10-week review, retrospective, and planning meeting (for the next 10-week PSI) that the 28 Scrum teams are involved in. PSIs consist of five 2-week sprints. Our goal was to evaluate the organization's Agile adoption during its earliest stages. We also wanted to obtain informal feedback to assess the reaction of the engineers (i.e., Agile team members) on the Agile adoption.

The second phase of the study took place over 3 days to observe the end of an iteration PSI 3-2 (i.e., the 3rd PSI meeting and 2nd 2-week sprint) meeting and the start of the following iteration PSI 3-3 meeting. This visit included 10 semi-structured 30-minute interviews with both engineers (e.g., developers, testers, Scrum masters etc.) and product owners (i.e., managers). The first set of interviews was conducted to assess the engineers' insights on the Agile adoption and their views on technical debt. The interview questions were threefold. The first set was demographic, the second set focused on their understanding of how Scrum impacted their individual projects and their perception of the Agile adoption. The third set was some general questions on managing technical debt. By analyzing the results of these interviews, we were able to refine our interview strategy and questions list for our third visit.

The main focus of the third visit was to carry out semi-structured interviews lasting 30 minutes with a central focus on technical debt. The target audience of the interviews was broadened to include hardware engineers, which collaborate with the software engineers. One aspect of this assessment is to determine the impact of the Agile adoption on the hardware team and whether technical debt is a concept hardware teams are familiar with. In addition, the pre-demographic questions from the second visit were sent in advance as an online questionnaire to make optimal use of the time. All but one participant responded to the online questionnaire.

Participation for the interviews was voluntary. The potential participants included all engineers within the division. Our industrial partner's management requested volunteers to give their feedback and many obliged. The interviews were conducted over a 2-day period from 9-5pm. During each interview, the first author took notes while the second author asked the questions.

As the focus of this work is on the technical debt aspect, we will focus only on the insights gained related to technical debt and not Agile adoption. The high-level research questions 1-4 were decomposed into the following interview questions:

Section 1: Technical debt

1. How would you define/describe technical debt?
2. How important is lowering technical debt in the organization? Why? Do you think your team does a good job addressing technical debt?

Section 2: Categories

1. Debts at the organization are typically classified as automation debt and infrastructure debt. How would you classify the debts that you address?

Section 3: Cost Estimation

1. Does your team incur debt intentionally? If yes, why? What are the benefits of doing so?
2. Do you record how much time you spend reducing debt?
3. If yes, how much time do you spend reducing debt?

Section 4: Prioritize/decision making

1. What type of debt is most difficult to address?
2. What methodology do you use to track debt?
3. How do you prioritize technical debt?
4. What are the impacts of technical debt for (1) the team (2) the customers (3) future modifications of the system?
5. How much time can a debt be on hold in the backlog?

1) Demographic data on participants

The participants first completed the online demographic survey prior to the interview. The demographic questions served to assess factors which might influence the answer of the engineers, namely work experience (within the organization and outside of the organization), educational background (i.e., computer science / software engineering, etc.) and amount of Agile training received. The questions from the online survey were as follows:

1. What system does your team work on?
2. What is your role in on the team?
3. What are your responsibilities?
4. How many people are on your team?
5. How many years of work experience do you have? (in and outside of the company)

6. How many years of Agile work experience do you have?
7. What academic degree do you have?
8. Are team members geographically distributed?
9. How often do members of the development team interact with stakeholders?
10. Did you receive any formal Agile training? Duration of training?

The 28 participants included product owners, Scrum masters and team members (i.e., developers, testers). The distribution of the team members can be found in Fig. 2.

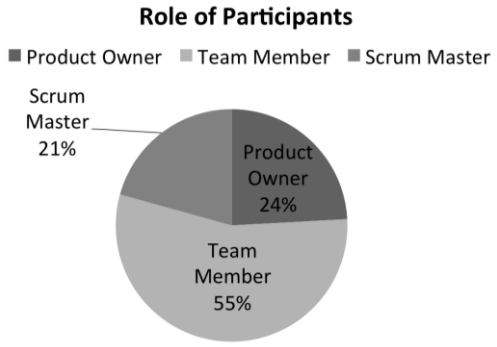

Fig. 2. Roles of participants

The tasks performed by the team members include coding, test automation, design, debugging, and other software development planning tasks (e.g., configuration management). The work experience of the team members ranges from 6 months to 30 years while the experiences in an Agile environment range from 4 months to 3 years. Only one participant received formal external Agile training while the others took part in an in-house training conducted by the organization's externally trained engineers for a maximum of 2 days.

C. Coding Scheme

Prior to the interviews, the questions were categorized according to themes. This process was the foundation for our coding scheme. Coding involves attaching labels or tags to pieces of text that are relevant to the different aspects of technical debt that are being researched. The process involves listening to the interview recordings, transcribing it and assigning these pre-identified codes to relevant pieces of text.

TABLE I. CODING SCHEME

Codes	Description
Definition	Words/phrases used to define/describe technical debt
Categories	Different types of technical debt
Causes and Impact	Causes - Motivations behind incurring technical debt Impact - Consequences of technical debt
Prioritization	Techniques/process to prioritize technical debt
Management	Tracking/managing/handling technical debt

The above categories were identified with the aim of providing insights and answers to the high level research questions. TABLE I. defines the coding scheme.

IV. RESULTS

This section summarizes the findings of the research with a predominant focus on interview results. The results are grouped according to the codes defined in the previous section.

A. Definition

Division management defines technical debt as infrastructure and automation debt. Infrastructure debt is the work that improves the team's process and ability to produce a quality product. Examples of ways to address infrastructure debt include refactoring, repackaging, and developing unit tests. Automation debt is defined as the work involved in automating tests of previously developed functionality to support continuous integration and faster development cycles.

We wanted to assess how the training impacted the participants' understanding of technical debt. Could they define it consistently? When asked to define what they understand by technical debt, most of the participants were familiar with the term.

One example definition:

- "People make bad decisions, not necessarily wrong but it would have been better if done in a cohesive way. Accumulation of technical debt occurs because there is no disciplined environment and speed – we want to respond to the customer faster, so we pull the technical debt credit card out and we will fix it later"

By "no disciplined environment", the participant explained that the development environment does not have a lot of rules. Techniques considered best practices by most (e.g., coding / design standards) are not strictly enforced. Speed refers to their need to get features released quickly.

Other examples include (grammatical edits in brackets):

- "It is a conscious decision to get things out quickly and plan to come back and address [any issues that arise]"
- "To do something in a hurry – not [necessarily] the right way."
- [When we] create bad software to get it out of the door and make money. Then we have to go back and fix it"
- "Our tests not [being] automated"
- "Something that will hurt you later. It is better to pay upfront. It is the lack of refactoring or anything that can be done more efficiently or anything that makes you slow down."

The majority of participants' responses indicate that technical debt is influenced by lack of time needed to properly design and code new features.

B. Categories

This code summarizes the types of debt encountered by the participants. Some example categories include:

- "Architecture, safety and consistency, packaging"

The practitioner described an example of safety and consistency debt as not initializing a variable or eliminating all compiler warnings.

- "Unit testing"

This refers to the amount code that has not been unit tested because the code is only modified in rare cases.

- "Automation debt, fix defects and bugs"

To 'fix defects and bugs' refers to all defect related debt. Several participants used this categorization. Automation debt includes test automation, which is one priority of the division to ease Agile adoption.

- "Test debt, bug debt"
- "Code design debt"

The participant mentioned that this type of debt was indicated by "plenty of code smells in the code base." These debts include creating unit tests and refactoring poor design choices.

When asked which type of debt is most difficult to handle, most participants mentioned architecture debt (i.e., debt related to refactoring or restructuring the system to encompass 'good design' principles) for the following reasons:

- "Changing system architecture requires lots of cooperation"

"Lots of cooperation" requires involving engineers from other teams in order to achieve the refactoring. The issues that require more cooperation are usually more difficult to address.

Another participant described their system as not being flexible enough when a particular feature impacts other features (i.e., when changing a portion of code requires touching other codes managed by different teams).

The responses from the interview indicate that the participants categorize technical debt mostly according to design, testing and defect debts.

C. Consequences

The motivations for incurring technical debt vary. One of the participants mentioned that the engineers do not know the "balance on the credit card and [keep] on charging". Management will decide when enough debt has been incurred, and this management decision is greatly influenced by customer needs. There are cases where software is released with known defects because of the fear of "breaking other things" while fixing a defect, which is a common occurrence in the software industry. Other more general responses for incurring technical debt intentionally include resource constraints such as a limited timeframe and the unavailability of developers to continue working on the feature.

Concerning the impact of technical debt, one participant mentioned, "if [the debt is] not solved for two years, it kills a project". He further elaborated that it is easier to re-write the software from scratch rather than go back and address the amount of technical debt that exists from a coding perspective. We could not conclude whether any projects were actually "killed" due to technical debt.

D. Prioritization

Customer requests are the predominant factor in determining if there are available resources to address technical debt. The product owner prioritizes technical debt based on customer needs. The second main factor of influence includes the severity of the debt, where in several cases, development work cannot continue until the debt is resolved. The debt is considered a blocking issue. The number of participants who quoted customers as the main factor to consider for prioritization was roughly twice as many as those who considered severity as the main factor for prioritization.

E. Management

There are different ways to handle infrastructure debt at the organization according to the participants: refactoring, reengineering and repackaging. Refactoring is "tidying up" the code, thereby increasing its maintainability without affecting its behavior. Reengineering is "shred it and re-do it", eliminating the code to rewrite from it scratch. Repackaging includes grouping "cohesive pieces that belong together with manageable dependencies. The aim is to simplify and architect the code so that we have only one copy and for consistency purpose, everybody uses that copy."

There are a few teams within this division that are solely dedicated to reducing technical debt. The technical debt reduction teams describe their work as "putting out little fires before it spread and at the same time, there are new fires in different areas." This analogy illustrates the fact that technical debt reduction is ongoing work. These teams address all types of infrastructure and automation debt.

In addition to the technical debt reduction teams, other teams also contribute to the reduction of technical debt, mostly by automating manual tests and fixing defects in their own code. However, they also pointed out that while they understand the benefits of lowering debt, some debts "stay in the backlog forever" if not a priority because the focus is on new features.

V. DISCUSSION

This section provides some insight on Agile adoption. Answers to the research questions identified earlier in the study are evaluated based on the results from the previous section.

In our first and second visits, we saw that shift to a more agile software development methodology was important because the previous methodology (more closely related to traditional waterfall) was no longer working. Some of the benefits of the switch based on informal discussion with engineers is that Agile is more focused, process-minded and the teams are aware of deadlines far in advance. Many of the participants interviewed highlighted increased visibility when referring to project management deadlines as to why they are pleased with the process. They know exactly what the priorities are for their current 2-week iteration and 10-week PSI. They analogize the shift from waterfall to Agile as "large irregular successes" to "small regular successes" and feel there is an increase in productivity now compared to the pre-Agile process. Management can know see what features are currently

being developed, and the release dates are more consistent. The developers get feedback on their work on a daily basis during the daily stand-up (daily Scrum) meetings, and they can request immediate help when problems arise. As they now work in multiple independent groups, they don't have to wait on others to do their work. Communication and collaboration has instilled a sharing culture among the teams as they are aware that the work accomplished by the other teams and know who to go to for help.

A. RQ1: How can technical debt be characterized to distinguish the impacts of certain types of debt?

With reference to Cunningham and Muller's description of technical debt [1][7], we noticed that the "definitions" given by the participants encompass more than speedy development, inability to manage maintenance costs, and handling software evolution.

According to the different taxonomies of technical debt presented in Section II, the participants described both intentional and unintentional debts. The engineers acknowledged that in order to deliver features to adhere to the 10 week PSI cycle, they had to compromise on some standard practices (e.g., complete test coverage, full unit testing, following known principles of good design) for the sake of the sprint schedule.

Several engineers recognized that the most difficult type of debt to reduce is architecture debt. Architecture debt can be described as debt incurred due to poor design decisions that affect the software structure and the interaction between objects.

As mentioned earlier, division management defines two types of debt for their process: infrastructure and automation debt. The participants described subcategories of these debts. It is difficult to state that there was a consensus among the different subcategories of debts addressed at the organization. This can be a potential problem when technical debt has to be lowered as management won't be able to distinguish between different types of debt and decide which one is more difficult to address than the other.

B. RQ2: What are the consequences of technical debt on the development process?

Technical debt was incurred to complete objectives and satisfy the customer. Developers feared acquiring more work if resolving some technical debts meant having to touch other related areas as a consequence of addressing the debt. Addressing the debt could thereby break other features and require more work from the developers.

There was only minimal insight obtained regarding the longer-term impact of technical debt. We feel that this is mostly due to Agile being newly introduced, and there is only limited observation of the impact of technical debts in an Agile context. Some practitioners feared that technical debt over the long-term has the potential to seriously hinder a project.

Despite the potential serious consequences of long-term technical debt, the practitioners are willing to take on debt to satisfy their short-term requirements.

Practitioners at our industrial partner are aware that incurring technical debt in the short term will help them towards achieving the objective of getting features released. However, they are unable to predict what would be the long-term impact of technical debt on the project. Based on these findings, we feel that the research community should focus on evaluating the risks of accepting certain types of debts. Such tools would be valuable in the software industry.

C. RQ3: How is technical debt addressed?

The participants mentioned three techniques to address architecture / design debt namely refactoring, repackaging and reengineering. While refactoring is a common technique mentioned in literature, reengineering and repackaging are less common.

The Agile Enterprise Team at our partner organization established several scrum teams whose only focus is to address technical debt (e.g., testing / test automation teams, defect and infrastructure teams). However, there were several participants who mentioned that some debts remain in the backlog for extended periods if they are not impacting the development of new features. The priority is to develop new feature and not to optimize existing ones.

Despite the fact that the organization is just beginning its Agile adoption, it has implemented some practices which seem to be beneficial to the organization. While the primary focus is producing new features, dedicated teams are assigned to reduce technical debt, and a majority of teams spend roughly 20% time in each PSI towards debt reduction. Such initiatives could be regarded as a best practice and adopted by other software development companies.

D. RQ4: How can technical debt be prioritized so that the most critical ones are addressed first?

In deciding which technical debts should be addressed first, none of the formal techniques described by Seaman et al. were used [9]. However, there was some coherence between the factors of influence mentioned by Snipes et al. and the way decisions to prioritize debt were made at the organization [11]. Severity and customer impact were considered important when deciding development priorities. Participants, however, did not mention any assessments based on expected effort to address the debt or any risks associated with a fix or the scope of testing.

In order to minimize the negative impact of technical debt, the most critical debts should be addressed first. It is important that a coherent mechanism be put in place to determine which debts are most critical based on its long-term impact. Currently, the organization is determining which debts are most critical based on whether the issue is "customer facing" or whether it prevents other work from being completed. The participants indicated that they knew how to handle debt prioritization based on the customer that is being affected. A process to determine factors influencing technical debt decisions should be agreed upon as it will become more difficult when the engineers have cases of technical debt that they think are equally important.

E. Limitations

There are some limitations to this study. First, the study was conducted with one industrial partner whose main focus was development of software for communication devices. Engineers working on other types of software systems were not considered. Second, the insights of the engineers were analyzed and interpreted by the researcher who may be biased by her perspective.

VI. Conclusions and Future Work

This study presents the insights and results of interviews carried out with engineers in one software development division of a mid-sized company. The goal was to understand how technical debt is characterized, addressed and prioritized as well as how each factor influences technical debt decisions. The industrial division under study recently (within a few months of the initial visit) adopted Agile. The insights provided are not from a mature Agile development group but a group that has had success quickly adopting and implementing Agile for around 250 developers and 28 newly formed Scrum teams. The insights relating to technical debt should be beneficial to other companies interested in adopting Agile practices. The large sample size of the group involved in this study allowed us to interview multiple Agile roles and provided actual practitioner insights which could be shared with the software engineering community.

The interview participants came up with several definitions and categories of technical debt based on the type of work they are responsible for. There was lack of consensus among the engineers related to technical debt terminology, which may come with time as the process matures. Other insights include:

- Further work is needed in order to evaluate the risks of taking on certain types of debt, both in the short term and long term.
- One effective debt management strategy includes having dedicated teams whose aim is debt reduction and also each team use about 20% of the PSI to focus on debt reduction.
- While prioritization of technical debt is greatly influenced by its impact on the customer and the severity of the debt, further study is needed to be able to determine which debts are more critical in the long term.

We would like to replicate this study at our industrial partner as the process matures and replicate with other industrial partners. Ultimately, our goal is to use these industrial findings with other empirical investigations of technical debt to develop a scheme to characterize various types of debt, evaluate the impact of these debts in certain contexts, and use this information to prioritize and plan technical debt reduction.

Acknowledgment

The authors would like to thank our industrial partner and their employees for their eager participation in this study.

References

[1] W. Cunningham, "The Wycash Portfolio Management System," *Addendum to the proceedings on Object-oriented programming systems, languages, and applications (Addendum)*, Vancouver, British Columbia, Canada, 1992, ACM, pp. 29-30.

[2] B. Curtis, J. Sappidi, and A. Szynkarski, "Estimating the Size, Cost, and Types of Technical Debt", *Managing Technical Debt (MTD), 2012 Third International Workshop on*, 2012, pp. 49-53.

[3] E. Lim, *A Balancing Act: What Software Practitioners Have to Say About Technical Debt*, N. Taksande and C. Seaman, Editors. 2012. p. 22-27.

[4] M. Fowler, *Technicaldebtquadrant*, 2009.

[5] T. Klinger, et al., "An Enterprise Perspective on Technical Debt," *Proceedings of the 2nd Workshop on Managing Technical Debt*, Waikiki, Honolulu, HI, USA, 2011, ACM, pp. 35-38.

[6] S. Mcconnell, *10x Software Development*, 2007.

[7] M. Muller, *Interview with Ipek Ozkaya (Sei) on Technical Debt, Agile and Architecture*, 2012.

[8] A. Nugroho, J. Visser, and T. Kuipers, "An Empirical Model of Technical Debt and Interest," *Proceedings of the 2nd Workshop on Managing Technical Debt*, Waikiki, Honolulu, HI, USA, 2011, ACM, pp. 1-8.

[9] C. Seaman, et al., "Using Technical Debt Data in Decision Making: Potential Decision Approaches", *Managing Technical Debt (MTD), 2012 Third International Workshop on*, 2012, pp. 45-48.

[10] C. Seaman and N. Zazworka. *Identifying and Managing Technical Debt*. 2011; Available from: http://www.slideshare.net/zazworka/identifying-and-managing-technical-debt.

[11] W. Snipes, et al., "Defining the Decision Factors for Managing Defects: A Technical Debt Perspective", *Managing Technical Debt (MTD), 2012 Third International Workshop on*, 2012, pp. 54-60.

[12] E. H. Stephen Chin, Walter Bodwell, Israel Gat, "The Economics of Technical Debt *", Cutter IT Journal*, vol. 23, no. 10, 2010.

[13] L. Williams and A. Cockburn, "Agile Software Development: It's About Feedback and Change*", Computer*, vol. 36, no. 6, 2003, pp. 39-43.