2014 6th IEEE International Workshop on Managing Technical Debt

Technical Debt and the Effect of Agile Software Development Practices on It – An Industry Practitioner Survey

Johannes Holvitie & Ville Leppänen
TUCS - Turku Centre for Computer Science
Software Development Laboratory &
Department of Information Technology
University of Turku
Turku, Finland
{jjholv, ville.leppanen}@utu.fi

Sami Hyrynsalmi
TUCS - Turku Centre for Computer Science
Software Development Laboratory &
Department of Management and Entrepreneurship
University of Turku
Turku, Finland
sthyry@utu.fi

***Abstract*—A major reason for the popularity of agile and lean software methods is their capability to function in resource scarce and requirement erratic environments. Both of these characteristics cause accumulation of technical debt, something that is the end result of either intentional or unintentional decisions. The ability of these methods to function with technical debt indicates that they contain components with inherent technical debt management capabilities. This study conducts a survey on industry practitioners to discover what is their level of technical debt knowledge, how does technical debt manifest in their projects and which of the applied components of agile software development—both processes and practices—are sensitive to technical debt. This paper contributes to the technical debt discussion by showing differences in assumed and indicated technical debt knowledge. Furthermore, components closest to implementation and its maintenance are perceived to have the most positive effects on technical debt management. Finally, the most encountered instances of technical debt are caused by architectural inadequacies, they are internal legacy, and increase in size as a result of continued implementation.**

I. Introduction and Motivation

The modern software development environment expects projects to cope with frequently changing requirements and optimally scarce resources while efficiently delivering complete software products or services. Few decades ago it was seen that traditional plan-driven methods of software development (c.f. [1]) would not cope within these premises [2]. This fact was emergent to a school of development methods which divided the implementation effort into smaller, manageable iterations that would pursue completion of partial software components, i.e. increments. This group of methodologies is generally referred to as the agile—and the lean—development methods and their formal definition can be seen to come in the form of a manifesto [3].

Agile development methods describe processes and introduce practices that solve a number of software development issues [4]. For example, agile methods tackle issues related to resource allocation for constantly changing requirements. While this matter has resulted in wide adoption for these development methods, projects using them do not go without their shortcomings. One reason for this is that they are still applied by individuals which makes them sensitive to subjective decisions. We as individuals make decisions based on the information that is available to us. When this information is not exhaustive, the end result can deviate from optimal. Similarly, we may perceive that an optimal solution is not required but rather that an adequate one should be fitted now to save us some time and money. The results of the two aforedescribed mechanisms represent the two types of technical debt introduced by McConnell [5].

The concept of 'technical debt' [6] captures the deviations that are present in a software product, how continued development affects them, and what is their effect onto the hosting development process. The involvement of individuals into development ensures that there is technical debt in each and every software development undertaking. The success of agile software development methods, however, indicates that the processes and practices they offer are capable of managing the technical debt within. While there is a plethora of work related to technical debt and its management (e.g. [7]–[9]), to the best of authors' knowledge, extant research regarding professionals' perceptions about the technical debt capabilities of existing agile software practices and processes is scarce. This paper addresses this gap by conducting an online survey for the professional software developers in Finland and received 54 applicable responses.

The results show that knowledge of the technical debt concept varies widely among the respondents. When an explanation of the concept is given, the respondents' confidence increases. The rise of respondents' confidence seems to indicate that the used technical debt type [5] and effect [10] definitions seem to be intuitive and practical. Nevertheless, there is a remarkable share of software development professionals who were not familiar with the concept.

We also show which agile software practices are felt to either reduce technical debt or enhance its management in development projects and which practices are perceived to do the opposite. Finally, the results reveal that almost nine out of ten technical debt instances reside in the implementation, they are caused by inadequacies in the architecture, and they are organization's internal legacy. This and capturing a low

978-1-4799-6791-9/14 $31.00 © 2014 IEEE
DOI 10.1109/MTD.2014.8

number of out-of-scope reasons, seems to indicate that the used technical debt cause listing, by Kruchten et al. [11], captures the space rather well.

The rest of the paper is structured as follows. Earlier survey studies regarding technical debt in software development are discussed in Section II. Section III formalizes the research questions for this study. Based on them, a survey is designed in Section IV. There are three consecutive parts to the survey that capture general information, used software development techniques, and perceptions about technical debt and encountered instances. Section V describes the survey implementation as a web based questionnaire and the process of data collection. Results are presented and analyzed in Section VI. Based on this, limitations and discussion is carried in Section VII. And finally, conclusions are drawn in Section VIII.

II. Related Work

Even though the term 'technical debt' was coined over twenty years ago by Ward Cunningham [6], research on the matter has been scarce up until the last few years (e.g. [7]–[9], [12]). This trend has also been emergent to a number of industry studies explaining technical debt in applied software development. Many of these studies, like the one in question, are conducted as practitioner surveys.

Klinger et al. [13] interviewed four experienced software architects regarding technical debt accumulation. They concluded that many times the decisions to incur technical debt come from non-technical stakeholders, stakeholders often have competing interests for which the best fit is usually technically non-optimal. Snipes et al. [14] conducted an interview on members of software change control boards regarding defect debt management. In this, they concluded that majority of the defect debt cost came from investigating and validating it, and that there are six major components that affect decisions on incurring/paying defect debt. Both the cost and decision component factors together influence how efficient the chosen strategy is in managing defect debt. Spinola et al. [15] collected a number of common phrases regarding technical debt into a 'folklore list' and surveyed practitioners to indicate with which they agreed and how spread the consensus was regarding them.

We found two survey studies regarding software development practices and singular instances of technical debt. Codabux and Williams [16] conducted a study to identify how technical debt should be characterized, what are its consequences on to the development process, what are its current management approaches and how should technical debt be prioritized for management. Respective to this, they found that developers considered their own taxonomy for technical debt to often be the best fit, dedicated teams as well as allocated maintenance tasks were considered for management, the technical debt consequences varied according to intentionality and stakeholder involvement, and finally customer needs were predominant while debt severity followed this for prioritization. This study was conducted within a large software organization using semi-structured interviews and questionnaires with open-ended questions. Finally, Lim et al. [17] interviewed 37 practitioners regarding how technical debt manifested, in which contexts, and what management practices were applied for it. The study found that 75% of participants were not familiar with technical debt, many said that it was incurred due to informed decisions and often due to competing concerns, the effects were mainly long-term and affected internal quality, and management was seen difficult as measuring non-uniform debt is a challenge.

In this study, we focus on distinguishing individual agile software development practices and processes by using a more close-ended approach than what the previously described surveys did; however, we aim to retain comparability by adhering to their structures where possible (e.g. capturing individuals' development background and the used verbal scales).

III. Research Questions

As per Section I, the success of agile development methods indicates that they are capable of managing or at least accommodating technical debt within the project. These development methods are defined through the processes they describe and the practices they use. In this study, a mapping of common agile processes and practices sensitive to technical debt is built by conducting a survey on industry practitioners. Two notable factors affect the aggregation of the mapping: individual's software development background and technical debt knowledge. From these subjective factors, we can only record the state of the former but we can improve the latter during the survey. That is, as part of the study we ensure that the respondents are exposed to and understand the definition of technical debt. We formalize studying these two factors and aggregation of the mapping in the following three research question groups.

First, we want to establish what is the individuals' software development background and how does this reflect in what he/she perceives his/hers assumed and actual technical debt knowledge to be. For this reason we formulate research question group *RQ1* as the following:

For an individual
- RQ1.1 *does work experience,*
- RQ1.2 *do used agile development practices, or*
- RQ1.3 *do associated project responsibilities*

correlate with what the respondent perceives his/hers assumed or actual technical debt knowledge to be?
- RQ1.4 *in which mediums has he/she seen or heard the term technical debt be used?*
- RQ1.5 *in which situations has he/she or his colleagues applied the concept of technical debt?*
- RQ1.6 *in which situations does he perceive the use of the technical debt concept as helpful?*

While capturing *RQ1*, we simultaneously ensure that the respondents are introduced to definitions given for technical debt and its effects (see Section IV-C). Relying onto ensured technical debt knowledge, we proceed to map which agile development practices does the respondent use and how does he/she perceive them to affect the project's technical debt. These form our second group of research questions (*RQ2*):

Are there certain software development practices for which
- RQ2.1 *their effect on technical debt is seen to be significantly positive, neutral or negative?*
- RQ2.2 *it is seen that they (do not) cover the team's or the project's development management needs?*
- RQ2.3 *it is seen that they (are not) able to cover the entire space of matters that require management?*

After *RQ1* and *RQ2*, we have established the fundamentals of technical debt for the targeted individuals in addition to querying the role of technical debt in their software development organization. Based on to this, the respondents are asked if they can distinguish singular instances of technical debt. Gathering information on these different manifestations of technical debt is formalized in our last research question group *RQ3*:

For a concrete instance of technical debt

- RQ3.1 *in which project component does it reside?*
- RQ3.2 *what are the causes for its emergence?*
- RQ3.3 *is it legacy?*
- RQ3.4 *is its size dynamic?*
- RQ3.5 *does its effects correlate with its size?*

The following section describes the survey designed to capture the previously introduced three research question groups as opinions of industry practitioners.

IV. Study Design and Background

The study in question aims to establish which agile software development processes and practices are sensitive to technical debt through surveying industry practitioners. This section describes the three consecutive parts of the survey designed to implement the study. The three parts are derived from the research questions introduced in Section III. Each of the following subsections discusses objectives for a single part and derives survey contents from supporting related work.

A. General Information

The first part of the study establishes the respondents' software development background. The questions focus on discovering basic details. The only purely personal information queried is the individuals' involvement in software development and it is recorded as a selection from three options: *under three years, three to six years or over six years of experience*. This is the first component to discovering *RQ1.1* and roughly divides the respondents into novices, experienced ones and veterans as per Salo and Abrahamsson used in [18].

Organizational details are queried at the project and at the company level. For the company level, basic information regarding size and the number of concurrent projects is recorded. Project level questions expand on the company ones and focus on the respondent's host project. If a person is working on multiple projects, the survey limits questioning to the one the respondent feels the most affiliated with. Project details queried include scoping the software product deliverable, the number of team members, and the development cycle times. What is also recorded is what project responsibilities does the respondent assume or to which ones is he appointed to in the project. Used responsibility classification is adopted from [19] and it is the first component to discovering *RQ1.3*.

B. Used Development Techniques

The second part of the study focuses on establishing which agile software development processes and practices are applied by the respondent's team or project. Time and other limitations forbid querying this with an exhaustive option list. As such, a compilation of the most common agile practices and processes was composed for this purpose. The list of practices (see Table I) was formed based onto the guidelines of the Extreme Programming (XP) method [20]. In XP, some practices are introduced as strategies, for these the same translation to practices was used as in [18]. Extreme Programming is a widely adopted [21], practice-heavy, agile software development method. A lot of XP practices are either adopted or their use is encouraged by other agile methods [22].

TABLE I. Practices of the Extreme Programming Method

Planning Game	Continuous Integration
Simple Design	40-Hour Week
Test-Driven Development	On-Site Customer
Refactoring	Coding Standards
Pair Programming	Open Office Space
Collective Code Ownership	

Another widely adopted and popular agile method is the Scrum [23]. Scrum is an example of a method which can be used concurrently with XP because it focuses on defining process components and leaves the practicalities open for choice [22]. The list of queried agile processes is based on an abstraction of the process components defined by the Scrum development method (see Table II).

TABLE II. Abstracted Process Components of the Scrum Method

Iteration Planning Meetings	Iteration Reviews/Retrospectives
Iterations	Daily Meetings
Iteration Backlog	Product Backlog

In querying the respondents about their projects' adoption of the aforementioned agile processes and practices, we used the five point verbal scale defined by Alreck and Settle [24]. In addition, to establishing the adoption levels, the respondents were asked to indicate if they considered the adopted group of processes and practices to be generally sufficient in covering their team's or project's development management needs and were they actually able to cover the entire space of matters that required management. The information recorded in this part of the survey is used to discover answers for research questions *RQ1.2*, *RQ2.2*, and *RQ2.3*.

C. Technical Debt

The final and largest part focuses on technical debt and it is divided into two sub-parts. The first sub-part establishes what is a respondent's knowledge about technical debt and ensures that it is in line with the definitions discussed in academic research. The second part relies on the respondent's refined knowledge and asks him/her to identify technical debt in his/her work and to classify the effects of adopted agile software development processes and practices on it.

The first technical debt sub-part starts by querying the respondent's prior knowledge about technical debt. Recording assumed knowledge is the second component for research questions *RQ1.1-1.3*. In addition to recording the general level of knowledge, the respondent is also asked to describe the concept. Further questions are made after this in order to establish where the respondent has seen or heard the term being used and in which situations. These questions record data for *RQ1.4* and *RQ1.5*. After establishing the base level of

knowledge, we ask the respondent to look at two definitions for technical debt. The first definition is by Steve McConnell [5] and it describes the processes for accumulating technical debt either intentionally or unintentionally:

> *The first kind of technical debt is the kind that is incurred unintentionally. For example, a design approach just turns out to be error-prone or a junior programmer just writes bad code. This technical debt is the non-strategic result of doing a poor job. In some cases, this kind of debt can be incurred unknowingly...*
>
> *The second kind of technical debt is the kind that is incurred intentionally. This commonly occurs when an organization makes a conscious decision to optimize for the present rather than for the future. "If we don't get this release done on time, there won't be a next release"...*

We chose not to provide our own definition here because we considered the one above to be prompt, intuitive, and exhaustive. Another definition was, however, required to describe the effects of technical debt. We used the one given by Brown et al. [7]. In this, an abstract definition for the possible effects is given by relating technical debt to financial debt:

> *The metaphor highlights that, like financial debt, technical debt incurs interest payments in the form of increased future costs owing to earlier quick and dirty design and implementation choices. Like financial debt, sometimes technical debt can be necessary. One can continue paying interest, or pay down the principal by re-architecting and refactoring to reduce future interest payments.*

After seeing both the definitions, the respondent is asked to indicate how close his/hers definition was to them. Indicated actual knowledge is the third component for research questions *RQ1.1-1.3*. Finally, for *RQ1.6*, the respondent is asked to indicate in which situations he/she would see the use of the technical debt concept beneficial.

Having ensured that the respondent understands the concept of technical debt, the second technical debt sub-part focus on concrete matters. Here it is established if the respondent perceives that there are components to his/her work that are affected by technical debt. This question is followed by the same agile practice (see Table I) and process (see Table II) lists as before. This time, the respondent is asked to indicate for each applied practice and process the overall effect it has on technical debt. The effect for each is recorded as an option chosen from the five point verbal scale: *very positive, positive, neutral, negative, or very negative effect*. The abstract term 'effect' is described to the respondent: if a practice or a process has a positive effect it "*can for example enhance technical debt management, lower its accumulation, or decrease its effects*". A negated definition is given for the negative options. Analyzing the captured process and practice data corresponds to studying the main research question *RQ2.1*.

After indicating if there is technical debt in their work, the respondents are asked if they can provide a more detailed description for a single component affected by technical debt. The detailed description captures in which high-level project phase does the component reside: *requirements elicitation and analysis, design, implementation, or testing* as described in [19]. The causes or the reasons for the technical debt's existence are captured as choices from *The Technical Debt Landscape* as presented by Kruchten et al. [11]. The component's origin is also of great interest and it is captured as possible legacy from an internal or external team or project (see Figure 8). Finally, the correspondent is asked to estimate the dynamics of both the technical debt instance's size as well as its perceived effects. Together, the component specific questions provide data for analyzing the third research question group *RQ3.1-3.5*.

V. Study Implementation and Data Collection

Three research groups where introduced in Section III to study the general level of knowledge industry practitioners have of the technical debt concept, which agile software development practices and processes are sensitive to technical debt and how do concrete instances of technical debt manifest. A survey was designed in Section IV to facilitate answering these research questions. This design was implemented as a web-based questionnaire that was sent to a predefined group of software development organizations.

The choice of a web-based questionnaire was made in order to increase response rate and minimize data handling errors [25]. The Google Forms platform (see *https://docs.google.com/forms/*) was used for the survey's construction, distribution and data collection, while the Bitly service (see *https://bitly.com/*) was used for link management and tracking. The final questionnaire contained 37 questions. From these, 34 were close-ended and 3 were open-ended questions. An answer was required to all but one question in the main part of the questionnaire. The optional technical debt component definition part contained six questions that required to be answered. All questions were organized and trialed so as to ensure the respondents' motivation [26]. Also, the entire survey was designed to take around ten minutes to answer, while opting to define a technical debt component would increase this by a couple of minutes.

The finalized survey was sent to a predefined group of software companies based in Finland. The set was built from companies for which a clear software product development role could be established: either the company belonged to a national level software development organization or its standard industrial classification [27] indicated software development activity. For the chosen group of organizations, if the preliminary investigation yielded direct contact details for all people working with software development, these individuals were approached directly. For organizations for which these details were not revealed, we contacted management and asked them to distribute the survey internally. Regardless of the approach, all people were informed about the survey with a cover letter that detailed the motives and goals of the study [26].

VI. Results and Analysis

The study in question surveyed industry practitioners to establish 1) what is the level of technical debt knowledge for them, 2) which common agile software development practices

and processes are perceived to be sensitive to technical debt, and 3) how does technical debt manifest in their work. Section III formalized studying these matters as research question groups, a survey was designed based on to these groups in Section IV and applied in Section V. In total, 507 individuals were approached in 153 companies and 80 responses were received. From these, 54 had completed the survey questionnaire. In this study, we focus on the answers from these 54 respondents (N=54). While our survey set consisted from software companies based in Finland, we recorded accesses to our survey also from India, Sweden, Greece and one other unidentified European country. The following subsections present results and analysis for the research question groups in Section III.

A. Individual Perceptions about Technical Debt – RQ1

Research question group *RQ1* focused on individual perceptions about technical debt. Figure 1 captures respondents' prior technical debt knowledge and their closeness to the presented definitions. These distributions were classified according to respondents' work experience (*RQ1.1*), applied software development techniques (*RQ1.2*) and assumed roles (*RQ1.3*). There was no significant difference between the distributions of these variables. As such, the most general one is presented here. From Figure 1 it is visible that perceptions about prior technical debt knowledge seem to be rather normally distributed with most respondents indicating it as being either good or adequate. After having seen the definitions for technical debt's types and its effects, the distribution changed. Now over 80% of the respondents indicated that their definition was either close or very close to the shown ones. It is also worthwhile to note that over 20% of the respondents indicated poor or no technical debt knowledge and this shifts to an even worse after having seen the definitions.

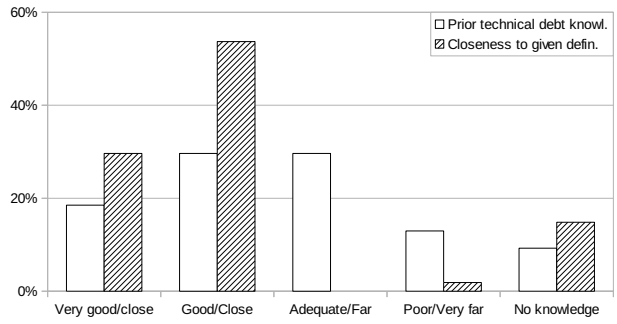

Fig. 1. Distributions for perceived prior technical debt knowledge and closeness to given technical debt definitions

The respondents were asked where they had either seen or heard the term technical debt used (Figure 2). As expected, occupation related areas are dominant and, surprisingly, over half of the respondents reported the term has been used in work meetings. On the other hand, 15% of the respondents reported never seeing nor hearing the term being used.

Finally, a mapping of common decision situations in development is shown in Figure 3. From this, it is visible that more than half of the respondents perceived the use of the technical debt concept useful in all presented scenarios. Current practice

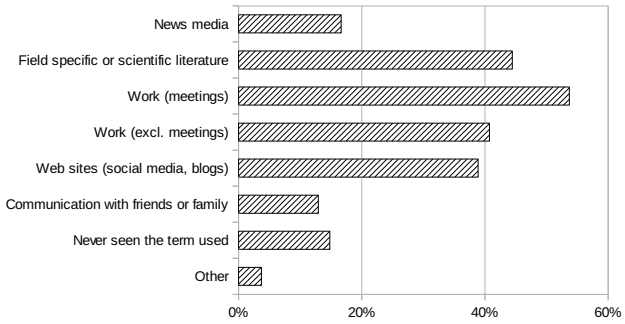

Fig. 2. Respondent technical debt usage in different mediums

however is different: from the same figure it is visible that only 35% of the respondents and their colleagues had applied the concept while a similar portion had never applied it. Noteworthy is also that more than 15% of the respondents perceived that use of the technical debt concept would not bring any gains to development decision making.

B. Agile Software Development Practices – RQ2

The main research question group *RQ2* focused on the capabilities and effects of common agile software development practices and processes. Figures 4 and 5 capture perceived effects on technical debt for discussed practices (see Table I) and processes (see Table II) respectively. What is visible from Figure 4 is that practices immediate to the product implementation (*Simple design, TDD, Coding standards, Refactoring, Continuous integration, Collective code ownership and Pair Programming*) are indicated as having positive or very positive effect on technical debt by over half of the respondents. *Coding standards* is perceived to have the most positive effect from all. Many practices are also perceived neutral to technical debt. Especially, over half of the respondents indicate *40-hour week* and *Open office space* as neutral. Finally, the widest effect distribution is recorded for *On-site customer* practice: 40% perceive this as neutral while both positive and negative effect have a 20% share.

No large differences are present in Figure 5 which captures perceptions about effects on technical debt for used agile pro-

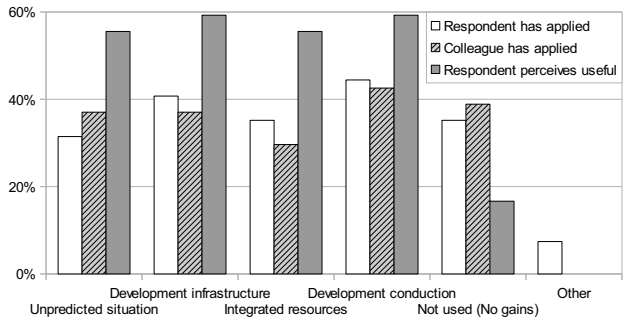

Fig. 3. Respondents' application of and perceived usefulness of applying the 'technical debt' concept for different development scenarios

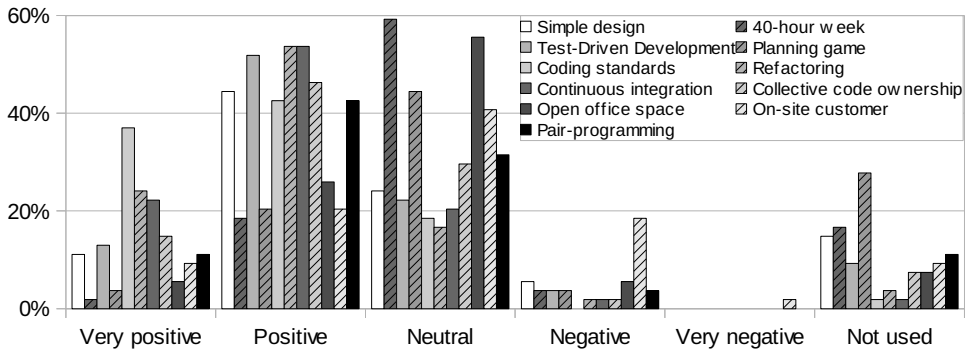

Fig. 4. Perceived effect of agile software development practices on technical debt

cesses. Opinions are less distributed here than in the practice chart (c.f. Figure 4). For all processes, circa two thirds from the respondents perceive them as either positive or very positive. *Iteration reviews/retrospectives* process is perceived to have the most positive effect from all. Somewhat surprising is that 6% of the respondents indicated that *Iterations* have a negative effect on technical debt.

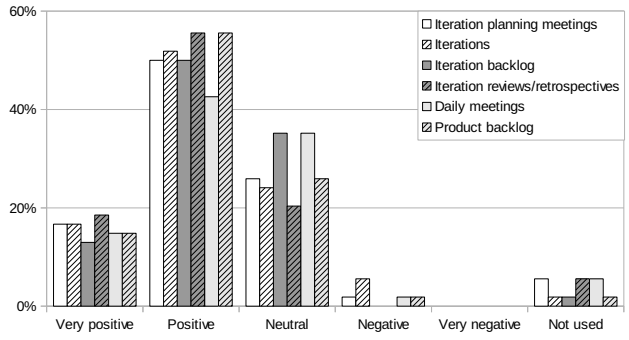

Fig. 5. Perceived effect of agile software development processes on technical debt

Related to the practices and processes, the respondents were also queried if the combination of agile techniques they used were adequate for the team's or the project's management needs (see *RQ2.2*) and were the techniques able to cover all aspects that require management (see *RQ2.3*). For singular practices, processes and their adoption rates, not a single combination could be identified for which the difference in their management or cover characteristics was statistically significant.

C. Manifestations of Technical Debt – RQ3

The last research question group *RQ3* concentrated on identifying instances of technical debt. While 93% of the respondents indicated that there are components to their work that are affected by technical debt, only 30% (N_{comp}=16) opted to describe an instance. While this number disallows making of statistically significant decisions, some interesting observations can still be made.

Figure 6 displays the distribution of technical debt instances as a function of project components (see *RQ3.1*). In general, a single technical debt component was indicated to affect a bit over two project components (avg. 2.3). Unsurprisingly, the most affected project component was the *Implementation* itself. More surprising was that both *Requirements/Analysis* and *Design* were indicated as more affected than the *Testing* component. The differences however are small between these components.

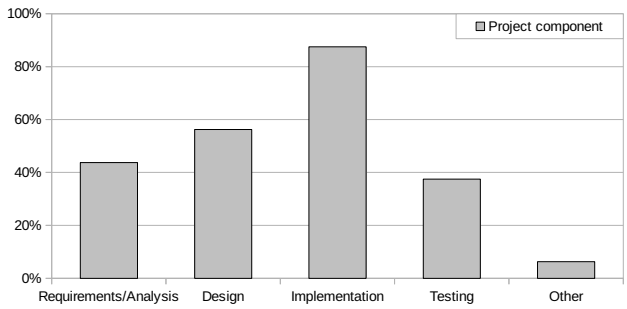

Fig. 6. Distribution of technical debt by project components

The causes were also recorded for each technical debt instances (see *RQ3.2*) and they are presented in Figure 7. For each instance a bit over four and a half causes were indicated (avg. 4.6) which is quite high considering that the pre-specified list consisted of a total of nine causes. Most frequently indicated causes were *Inadequate architecture* and *Inadequate documentation*. Indicated as a cause in 13 and 11 cases out of 16 respectively. Less frequent were *New features are required* at 4 incidents and *Defects or bugs* at 5 incidents. All in all, it seems that obfuscation of implementation structures is seen to cause much more technical debt than incomplete functionality.

RQ3.3 was interested in the origins of technical debt prone components and the distribution for this is captured in Figure 8. In three cases out of four, the technical debt is indicated to be legacy. From this, 90% is indicated to be internal legacy.

For *RQ3.4* the respondents were queried if they perceived that continued development for the components affected the size of the technical debt in them. The results are depicted in Figure 9. Over half of the respondents indicated an increase in size but no one indicated it to be a large one. A fourth

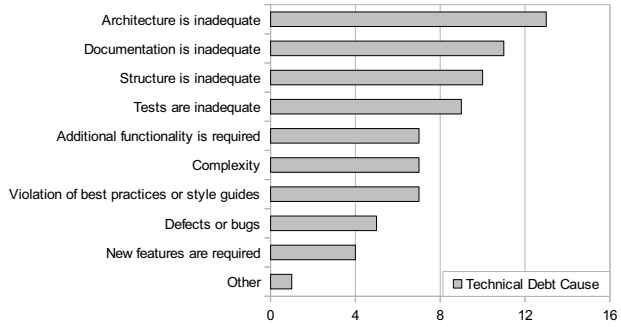

Fig. 7. Indicated causes for technical debt instances

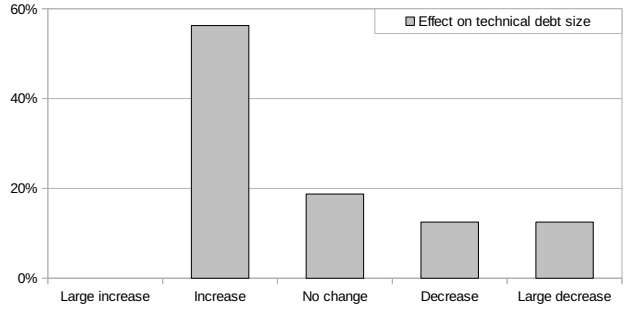

Fig. 9. Effect of continued development on size of technical debt instances

indicated either a decrease or a large decrease in size while one fifth indicated no change in size.

Finally, related to the previous paragraph, respondents were asked if they saw a correlation between the size of the technical debt component and the magnitude of effects it caused on development (see *RQ3.4*). From Figure 10, we see that in almost 90% of the cases the effects magnitude is seen to correlate with size. From this, in a bit over 60% of the cases the respondents were however unable to specify in any detail, how the size correlated. For two cases, respondents indicated that size and effects magnitude were not connected.

VII. Discussion and Limitations

The research questions and the implications of the study are discussed in the following. Regarding research question group *RQ1*, practitioners are quite familiar with the concept of technical debt but they assume less of themselves. This can be one reason for why technical debt is generally applied in 20% less cases than it is seen useful for. While the general outlook regarding the concept's increased use is positive, there is still a large portion of over 15% that have no knowledge of the term and have never seen or heard it being used.

For the second research question group *RQ2*, we note that agile software development practices which safe guard the state of the implementation itself are considered as having the most positive effect on technical debt and its management. Similarly to other discussed surveys, the competing interests of various stakeholder groups are shown for the practices as well. The on-site customer practice has the widest distribution. While it is possible that this practice actually has a negative effect on technical debt, we speculate that another option is that application of this practice has resulted in developers feeling intimidated by their less technically orientated customers. Less can be said about the surveyed agile software development processes. While for all processes a generally positive effect is indicated, the iteration review/retrospective has the highest score. From the perspective of competing stakeholder interests, this is an interesting process: the customer is often on-site, but the floor is generally received for developers, who use it to communicate about technical matters. It seems that limited, controlled customer involvement works best, while unlimited does not—or its least effective.

Finally, the third and last research question group *RQ3* focused on instances of technical debt. In almost nine cases out of ten, the project component affected by technical debt was the implementation itself. This provides further reasoning for why the practices focusing on implementation state maintenance (see Figure 4) are perceived to have the most positive effect on technical debt. Of interest is also the fact that several respondents indicated that the root cause for the project components' technical debt was inadequate architecture. This is partially explained by the fact that in over half of the cases the implementation design is considered to be prone to technical debt. However, further studies with a larger sample are required to confirm this. Finally, we observed that in three

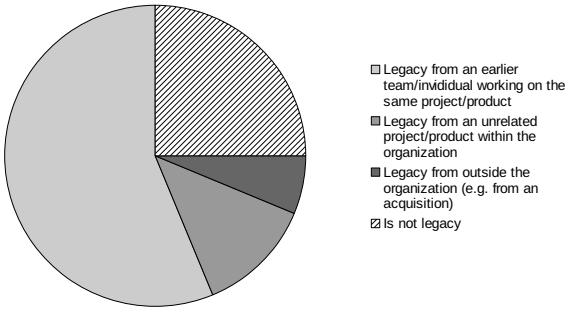

Fig. 8. Origins of components carrying technical debt

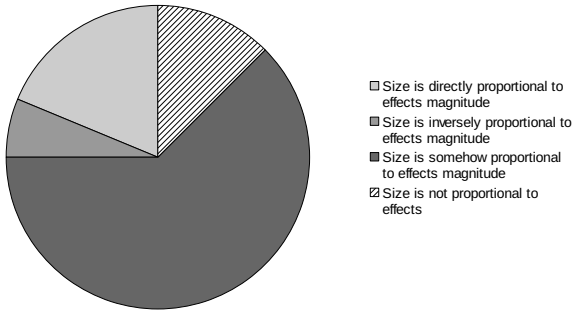

Fig. 10. Proportionality of technical debt instance size to its effects magnitude

cases out of four the components were legacy, their size altered as development was continued and the effects were proportional to their size. All these matters speak for integrating new and enhancing existing technical debt management approaches in addition to emphasizing knowledge management for the implementation.

Similarly as with other studies, also this research is limited by certain issues. First, the data was gathered from software development companies based in Finland and, thus, the results might be biased by the county's culture. Second, it is possible that only those developers and companies, who were highly interested about the technical debt concept and agile software development, answered to the questionnaire. This might bias the results towards a more positive view of technical debt knowledge. However, over 20% of the respondents were not familiar with the concept and thus we assume that this positive bias is not significant. Third, the questionnaire was completely anonymous and it is possible that a major part of the respondents represent the same company. However, even in this case, the results would represent the real figures as the majority of development is done in larger companies.

Nevertheless, further work is needed to replicate the results in other countries and development cultures. Based on to the findings presented in this and upcoming studies, we intend to conduct studies to investigate if similar results can be attained through more quantitative measures of software development.

VIII. Conclusions

This paper studied the perceptions of Finnish professional software engineers to the concept of technical debt. The results show that there are variations to individuals' technical debt knowledge, but a majority perceives them to be close to the definitions discussed by research. While the concept is applied in practice, it is still under utilized in many possible occasions. Regarding agile software development practices, techniques that retain the implementation state are considered to have the most positive effect on technical debt and its maintenance. Structured and limited customer involvement carries a similar effect, while perceptions regarding the on-site customer practice were more dispersed. Additionally, we noted that most instances of technical debt affect the project's implementation and they were caused by inadequacies in the architecture and implementation structure. Finally, many of the instances were organization's internal legacy, their size was effected by continued development and the effects caused by these instances were often proportional to the size of the instance. Further studies on alternate software development cultures are expected to provide further reasoning for the results presented herein.

References

[1] W. W. Royce, "Managing the development of large software systems," in *proceedings of IEEE WESCON*, vol. 26, no. 8. Los Angeles, 1970.

[2] D. Leffingwell, *Scaling software agility: best practices for large enterprises*. Pearson Education, 2007.

[3] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries *et al.*, "Manifesto for agile software development," 2001.

[4] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.

[5] S. McConnell, "Technical debt," *10x Software Development Blog,(Nov 2007). Construx Conversations. URL= http://blogs. construx. com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx*, 2007.

[6] W. Cunningham, "The WyCash portfolio management system," in *Addendum to the proceedings on Object-oriented programming systems, languages, and applications (OOPSLA)*, vol. 18, no. 22, 1992, pp. 29–30.

[7] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya *et al.*, "Managing technical debt in software-reliant systems," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, 2010, pp. 47–52.

[8] I. Ozkaya, P. Kruchten, R. L. Nord, and N. Brown, "Managing technical debt in software development: report on the 2nd international workshop on managing technical debt, held at icse 2011," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 5, pp. 33–35, Sep. 2011.

[9] P. Kruchten, R. L. Nord, I. Ozkaya, and J. Visser, "Technical debt in software development: from metaphor to theory report on the third international workshop on managing technical debt," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 5, pp. 36–38, 2012.

[10] C. Seaman, Y. Guo, C. Izurieta, Y. Cai, N. Zazworka, F. Shull, and A. Vetrò, "Using technical debt data in decision making: Potential decision approaches," in *Managing Technical Debt (MTD), 2012 Third International Workshop on*. IEEE, 2012, pp. 45–48.

[11] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice." *IEEE Software*, vol. 29, no. 6, 2012.

[12] ——, "4th international workshop on managing technical debt (mtd 2013)," in *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 2013, pp. 1535–1536.

[13] T. Klinger, P. Tarr, P. Wagstrom, and C. Williams, "An enterprise perspective on technical debt," in *Proceedings of the 2nd Workshop on Managing Technical Debt*. ACM, 2011, pp. 35–38.

[14] W. Snipes, B. Robinson, Y. Guo, and C. Seaman, "Defining the decision factors for managing defects: a technical debt perspective," in *Managing Technical Debt (MTD), 2012 Third International Workshop on*. IEEE, 2012, pp. 54–60.

[15] R. O. Spinola, A. Vetro, N. Zazworka, C. Seaman, and F. Shull, "Investigating technical debt folklore: Shedding some light on technical debt opinion," in *Managing Technical Debt (MTD), 2013 4th International Workshop on*. IEEE, 2013, pp. 1–7.

[16] Z. Codabux and B. Williams, "Managing technical debt: An industrial case study," in *Managing Technical Debt (MTD), 2013 4th International Workshop on*. IEEE, 2013, pp. 8–15.

[17] E. Lim, N. Taksande, and C. Seaman, "A balancing act: what software practitioners have to say about technical debt," *Software, IEEE*, vol. 29, no. 6, pp. 22–27, 2012.

[18] O. Salo and P. Abrahamsson, "Agile methods in european embedded software development organisations: a survey on the actual use and usefulness of extreme programming and scrum," *Software, IET*, vol. 2, no. 1, pp. 58–64, 2008.

[19] B. Bruegge and A. A. Dutoit, *Object-oriented software engineering; conquering complex and changing systems*. Prentice Hall PTR, 1999.

[20] K. Beck and C. Andres, *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2004.

[21] L. Vijayasarathy and D. Turk, "Agile software development: A survey of early adopters," *Journal of Information Technology Management*, vol. 19, no. 2, pp. 1–8, 2008.

[22] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods: Review and analysis," 2002.

[23] K. Schwaber and M. Beedle, *Agile software development with Scrum*. Prentice Hall PTR Upper Saddle Riverˆ eNJ NJ, 2002, vol. 18.

[24] P. L. Alreck and R. B. Settle, *The survey research handbook*. Irwin Homewood, IL, 1985, vol. 2.

[25] P. M. Nardi, "Doing survey research: A guide to quantitative methods," 2006.

[26] A. Bryman and E. Bell, *Business Research Methods 3e*. Oxford university press, 2011.

[27] Statistics Finland, "Standard industrial classification TOL 2008," *Based on NACE 2008, the European Union Classification of Economic Activities*, 2008.