Technical Debt in Agile Development

Report on the Ninth Workshop on Managing Technical Debt (MTD 2017)

Francesca Arcelli Fontana
Università degli Studi di Milano-Bicocca
Milano, Italy
arcelli@disco.unimib.it

Wolfgang Trumler
Corporate Technology, Siemens AG
Erlangen, Germany
wolfgang.trumler@siemens.com

Paris Avgeriou
University of Groningen
Groningen, Netherlands
paris@cs.rug.nl

Alexander Chatzigeorgiou
University of Macedonia
Thessaloniki, Greece
achat@uom.gr

Clemente Izurieta
Montana State University
Bozeman, MT, US
clemente.izurieta@montana.edu

Robert L. Nord
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, US
rn@sei.cmu.edu

DOI: 10.1145/3127360.3127372
http://doi.acm.org/10.1145/3127360.3127372

ABSTRACT

We report on the Ninth International Workshop on Managing Technical Debt, collocated with the 18th International Conference on Agile Software Development (XP 2017) in Cologne. The technical debt research community continues to expand through collaborations of industry, tool vendors, and academia. The theme of this year's workshop was on technical debt in agile development. Presentations and discussion centered on the topics: technical debt at the code level, architectural technical debt assessment, agile approaches and their impact on technical debt management, and selling the business case of technical debt management.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – restructuring. Metrics – complexity measures, product metrics, process metrics. Software Architectures

General Terms

Management, Measurement, Design, Economics, Standardization, Agile Development.

Keywords

Technical debt, software economics, software quality, software evolution, software analytics, agile development.

1. INTRODUCTION

Researchers have met regularly since 2010, in the workshop series on Managing Technical Debt (MTD), to further study and better define the concept of technical debt and its applicability to software development. In addition, research focusing on technical debt has started to be part of the work discussed in the main tracks of major software engineering conferences. In 2016 there was a Dagstuhl seminar [1] focusing on advancing research on managing technical debt. Previous workshops had made progress by creating an initial landscape for scoping technical debt [2], establishing key principles of what constitutes technical debt and what does not [3], establishing how different lines of research in software engineering form the basis of technical debt research and starting the beginnings of a research roadmap [4]. This ninth workshop focused on topics related to technical debt management and addressing technical debt on the code level under special consideration of agile approaches. Therefore, one of the key questions was whether agile approaches support the removal of technical debt or if they foster their introduction due to a misaligned understanding of agile development.

2. CONTRIBUTIONS

The workshop began with a short introduction by Francesca Arcelli and Wolfgang Trumler on how far technical debt research has come through this series of workshops and a description of the major milestones achieved by the MTD community. The timeline demonstrates the impact of MTD research and the growing interest around this series of events covering this research area. The interest in technical debt research and its application in industry grew to a level beyond the limits of a workshop leading to the decision that the workshop will become a full conference in 2018.

The introduction and welcome was followed by a keynote given by Eltjo Poort of CGI[1], Netherlands. His talk was about "Selling the Business Case for Architectural Debt Reduction". Eltjo proposed to talk to the stakeholders in their language, which implies a switch in the architect's communication style from technical aspects to risk and cost.

Business stakeholders typically do not understand why development teams need to spend additional effort in refactoring activities to reduce technical debt, especially when it happens after the software has already been presented or shipped. They cannot see the direct impact on the project, thus favoring additional features over refactoring activities. To quantify technical debt by identifying the risk exposure related to certain functionality or features helps the business stakeholders understand the need for regularly planned refactoring. Furthermore, this risk-based approach helps to identify the technical risk of the project and to plan for mitigation strategies to assure project success.

The workshop continued with one session in the morning chaired by Jean-Louis Letouzey. The following papers were presented:

- Antonio Martini and Jan Bosch, *"The Magnificent Seven: Towards a Systematic Estimation of the Technical Debt Interest"* [5]
- Antonio Martini, Simon Vajda, Mohamed Abdelrazek, Allan Jones, Rajesh Vasa, John Grundy, and Jan Bosch ,*"Technical Debt Interest Assessment: from Issues to Project"* [6]
- Jesse Yli-Huumo and Kari Smolander, *"Changes and challenges of technical debt and its management during ongoing digital transformation*" [7]
- Theodoros Amanatidis, Alexandros Chatzigeorgiou, Apostolos Ampatzoglou, and Ioannis Stamelos, *"Who is Producing More Technical Debt? A Personalized Assessment of TD Principal"* [8]

[1] https://www.cgi.com/

The second session in the afternoon was chaired by Klaus Schmid with the following papers:

- Terese Besker, Antonio Martini, Jan Bosch, and Matthias Tichy , *"An investigation of Technical Debt in Automatic Production Systems"* [9]
- Thorsten Haendler, Stefan Sobernig, and Mark Strembeck, *"Towards Triaging Code-Smell Candidates via Scenarios and Method-Call Dependenciesz* [10]
- Natthawute Sae-Lim, Shinpei Hayashi, and Motoshi Saeki , *"Revisiting Context-Based Code Smells Prioritization: On Supporting Referred Context"*[11]
- Sofia Charalampidou, Apostolos Ampatzoglou, Alexandros Chatzigeorgiou, and Paris Avgeriou , *"Assessing Code Smell Interest Probability: A Case Study"*[12]

3. WORKING SESSIONS

The MTD workshop series has traditionally reserved the last portion of the meeting to lively discussions that engage all attendees. This year three topics were identified: architectural technical debt assessment, agile approaches and their impact on technical debt management, and selling the business case of technical debt management.

3.1 Architectural technical debt assessment

The discussion chaired by Alexandros Chatzigeorgiou and Francesca Arcelli was centered on the question whether technical debt (TD), in particular Architectural TD (ATD), can be assessed by means of methods and tools which eventually can support decision making of software architects. The following issues and challenges have been identified:

- Tools such as static code analyzers are by definition limited to identifying existing inefficiencies; which already reside in the code. They are unable to suggest meaningful opportunities for improving code quality with respect to forthcoming changes. Ideally, tools should not extract long lists of smells but provide hints to developers and architects so as to assist them in facilitating further software evolution and guide developers/architects on how to optimally reach a certain goal (e.g., the transition to a new database, adoption of a new framework/library).
- To this end it would be valuable to identify commonly recurring instances of ATD, classify them into well-defined and identifiable categories, which should be as project-neutral as possible. These instances of ATD would correspond to the black (invisible TD with negative value) and the yellow (hidden architectural features with a positive value) 'boxes' in the architecture backlog coloring scheme, according to Philippe Kruchten (see "What Color is Your Backlog" [9]). Setting up a repository of such architectural features, smells and decisions is of utmost importance to drive the development of efficient tools that can identify/quantify ATD.
- Tools for the identification/quantification of TD cannot rely on static source code analysis only. Multiple indicators should be exploited by analyzing all sources of information around a software project (including past decision logs, documentation, issues, change history, etc.). The ideal tool should act as a 'project cockpit' offering several complementary views of TD items, along with their associated risk.
- Identifying TD items without anticipating future scenarios for the evolution of a software system might render TD management useless. Capturing potential axes of change in the form of specific scenarios (which would express the functionality of the system in future versions or represent anticipated changes in architecture) is essential to steer TD identification and mitigation, aiming at facilitating these anticipated changes (scenario-based TD). Although the exact form of these scenarios cannot yet be specified, ideally, they should resemble the scenarios and test cases; which already exist for a given system.

Conclusion: The identification and quantification of TD should be elevated from the level of extracting long lists of low-level code smells to the level of suggesting architectural opportunities that have the power to facilitate future change scenarios.

3.2 Agile approaches and their impact on technical debt management

The guiding question of the discussion chaired by Wolfgang Trumler was on how agile approaches influence technical debt and whether they support removal or rather introduction of technical debt.

During the discussion three main topics were addressed, which have an important impact on technical debt in environments applying agile practices and methodologies: (1) the notion of value, (2) implementation-related aspects, and (3) important development practices.

3.2.1 The notion of value

Delivering customer value is at the heart of agile approaches. In many companies a strong focus on pure customer value has led to a misunderstanding of what value really is. It is often directly translated into delivering features to the customer. As a result, activities related to the reduction of technical debt are not valued high enough. Even worse, the strong focus on delivering features as fast as possible fosters the introduction of technical debt as there is not enough time to clean up the software and to conduct necessary refactoring.

Some companies started to introduce refactoring or stabilization sprints to address these issues and to make the need for these activities more obvious and transparent to the organization. In this case there are sprints without "direct customer value" and they are always in danger of being discarded, especially when development falls behind schedule, which typically happens because development teams have no time to clean-up their code and refactor the architecture to be prepared for upcoming features. The noticeable effect is an efficiency reduction over time by a decrease in development velocity.

However, there is a wider range of customers to a software product than just the end-customer. For example the operations team, the quality assurance (QA) department, and the developers, who may have to maintain the software for several years. If they were treated in the same way as end customers, all sprints would deliver value and the chances that technical debt would be addressed regularly to keep a constant delivery pace would be much higher.

3.2.2 Implementation-related aspects

In his book "The Lean Startup" [14] Eric Ries describes the "Build-Measure-Act" cycle. The fundamental idea is to implement a feature (basically a hypothesis) very quickly, and to put it in front of the customer to validate the hypothesis.

The problem related to this approach is that development teams might create a mess while going fast. However, the development team should not create something hard to live with afterwards. If the hypothesis has been invalidated, the changes to the code base can just be discarded. Otherwise the code has to be cleaned up and the architecture has to be refactored to immediately pay down the principal of the technical debt taken while going fast. Therefore, the development teams must not create poor code or untested features, in order to keep the effort for cleaning-up acceptable.

We call this the GMC-cycle (Go fast, Measure, Clean-up) where technical debt is deliberately assumed in order to be fast, but immediately paid back after the hypothesis has been validated. This approach can be used as a business model on top of the Build-Measure-Act-cycle. Ask for a small amount of money to create a minimal viable solution as fast as possible. When the customer is happy with the presented solution he or she has to spend further money to reduce technical debt as a precondition to add new features.

Another aspect towards the reduction of technical debt is that there should always be a good reason to remove TD. For example, analyzing a repository could give valuable hints. The number of check-ins in one module can be an indication of where to focus TD removal activities, or when changes are often spread across multiple modules for a check-in. This could be an indication of architectural debt because many different modules have to be touched to introduce a new feature.

Another issue was on how to approach design to keep technical debt as low as possible. On one extreme, there is no upfront design required when applying agile practices. This might be true for some very experienced engineers but not for the average developer and even harder with growing team size. If the developers lack a clear architectural vision, chances are high that they may make unsuitable design decisions jeopardizing the architectural corner stones. Therefore, the experience of the team members is a key success factor in the management of technical debt and at the same time, so are the decisions about the amount of upfront design required to provide guidance to the development team.

3.2.3 Development practices

During the transformation to agile approaches, many companies focus on the process-related part, overlooking or underestimating the benefit and value of the agile practices that build the foundation of iterative software development.

The original intention of agile approaches, especially from XP [15], was to deliver value to the customer by creating high quality software while at the same time reducing stress for the development team. This is achieved through short iteration cycles and by applying practices like Pair Programming, Test Driven Development, automated tests on all levels and continuous integration. Although the first and the last of the practices, short iteration cycles and continuous integration are employed by many teams switching to an iterative development style, the others are often neither trained nor practiced. However, these practices provide a huge benefit especially in the case of technical debt management where constant refactoring takes an important role, which is not possible without a testing safety-net.

3.3 Selling the business case of TD management

The third breakout group chaired by Paris Avgeriou was tasked with discussing how to sell the business case of managing technical debt, in accordance with the morning keynote talk that revolved around the same subject. Specifically, the keynote included several useful tips and ideas on how to convince management stakeholders to invest in the management of technical debt, mostly by focusing on the costs and risks. The group discussed these ideas and elaborated further based on industrial experiences of the participants. As a follow-up, the group also discussed the current state of practice in technical debt management in the different companies that participants represented.

3.3.1 Selling the business case

The participants agreed that the key to selling the business case is to be able to explain the return on investment in technical debt management. Both costs and benefits should be explicit, and of course benefits should outweigh the costs. The group discussed four different ways of selling the business case. The first was inspired by the keynote as several participants confirmed that presenting imminent risks is a very effective way to argue the case of technical debt management. Risks are quantifiable, so they provide some figures to base decisions on. They are also commonly used in stakeholder meetings, particularly those involving both technical and business stakeholders.

A second way that was put forward was simply to let the consequences of unmanaged technical debt speak for themselves (in simple terms, let the development team “feel the pain” of maintaining software with technical debt). Such consequences include the development team getting stressed, the development velocity being reduced, business goals not being met and generally projects running into trouble. Even when such problems are not very pronounced at a given time, they can still act as warning signs, such as when the development velocity is not very slow, but it is still slower than the competition. Identifying technical debt as the root cause of such issues is a solid argument to address technical debt. In certain companies, simply asking the development team to score the “pain felt” on a scale for example from 1 to 5 is enough to gauge the amount of technical debt and the urgency to resolve it.

A third way discussed in the group was to use language that business stakeholders understand. Arguing about low cohesion and high coupling might not be very popular with those stakeholders; but terms like debt and interest can be more persuasive. In addition using some simple examples from the system (e.g., code clones) and explaining their impact is important. In this case, it is important to have these arguments coming from technical stakeholders with credibility. Management is usually more receptive to the latter. It is equally important to find a sponsor in the organization that is willing to support investing in technical debt management.

The fourth way that was discussed is more of an anti-pattern: *it usually does not work so effectively to show figures from the source code analysis tools that measure technical debt and related qualities*. On the one hand, such figures expressed in person-months or dollars/euros are very simple. On the other hand not many people understand what they mean or how to interpret them or how accurate they are. The algorit hms and analytics used to calculate such figures are often complicated and don’t make much sense to business stakeholders. Some of them are highly customizable which makes the calculation process even less credible as it appears that tweaking some paramete rs can have drastic impact on reducing or increasing the calculated technical debt.

However there seems to be an exception when it comes to presenting features from code analysis tools. Some of the participants reported that their company’s management exhibit high technical maturity and most, if not all managers have a technical background. Of course, in those cases, such managers understand technical terms and metrics related to software complexity and they do value and support maintenance activities. They are able to customize and understand the output of the used tools and thus selling the case of technical debt becomes much easier.

3.3.2 State of practice

In most cases, companies use code analysis tools to identify, measure and manage their technical debt. This has certain limitations as mentioned in the previous subsection: such tools tend to make assumptions that are not explicit or clear even to technical stakeholders. Therefore the output is not precisely understood and its credibility is somehow compromised. In a few cases, participants reported the use of modeling or generic engineering tools (e.g., LabVIEW) that are able to manage technical debt at the design level. These are clearer when it comes to the performed analysis and provide more precise metrics. However such cases seem to be the exception.

The participants expressed some wishes and requirements for an ideal technical debt management tool. Clearly they would like an explicit analysis model and one that is customizable to the individual context of not only the organization but of the actual project. Furthermore the tooling should somehow represent the consequences of not managing technical debt, for example in terms of future maintenance costs or potential risks that can endanger a project. In addition an ideal tool should be able to show the gap between the system as-is and the system optimized to minimize technical debt; this difference would help stakeholders make explicit decisions on prioritizing and repaying technical debt items. Finally, the tool should hold a backlog of technical debt items ideally in relation to other forms of backlogs such as features to be implemented, bugs to be fixed or architecture-enabling elements. Generally the tool is expected to visualize technical debt in a way that is comprehensible to different types of stakeholders (inspired for example from tools like CodeCity or CodeScene).

Tooling in itself is useful but not enough to make a disruptive change in how industry manages technical debt. The group discussed that it requires organizational changes to achieve that. The organization culture needs to embrace the values of transparency and visibility. Technical debt needs to be explicit and well communicated and not 'hidden under the carpet' in order to show improved numbers or shift the problem to a different unit. The decisions to incur technical debt need also be well documented and accompanied by a rationale to make sure that future decisions are based on them instead of inadvertently violating them. Finally the group proposed to encourage a type of "boy scout mentality," for example in terms of developers leaving the code at a better state than they found it. This may sound idealistic but participants reported that it works in practice.

Organizations also need to be more resolute about technical debt and how to manage it. There are stakeholders who become aware of the concept and its importance but that awareness is sometimes rather superficial. There needs to be more education for different types of stakeholders to raise the awareness level. Furthermore the group suggested the repaying technical debt is often difficult as resources are scarce and are invested with an outlook on future features. Therefore organizations should focus on the future prevention of technical debt.

4. CONCLUSIONS

By holding this workshop together with XP, the workshop received a great deal of attention from leading software researchers and practitioners interested in exploring theoretical and practical techniques that manage technical debt within iterative and agile software development environments. The discussions during the working sessions generated interesting points for future investigation.

The workshop closed with the announcement that the MTD workshop will evolve into a two day working conference focusing on technical debt, collocated with the International Conference on Software Engineering in Gothenburg, Sweden in May 2018 (techdebtconf.org).

5. DISCLAIMER

The views and conclusions contained in this document are solely those of the individual authors(s) and should not be interpreted as representing official policies, either expressed or implied, of the Software Engineering Institute, Carnegie Mellon University, the U.S. Air Force, the U.S. Department of Defense, or the U.S. Government.

6. ACKNOWLEDGMENTS

Our thanks to the organizers of XP 2017, XP Workshop Chairs, and the keynote speaker Eltjo Poort. We would also like to thank all authors, session chairs, presenters, and attendees. Their contributions were significant in helping move forward in MTD's research agenda.

7. REFERENCES

[1] *Managing Technical Debt in Software Engineering*. Dagstuhl Reports 6 (April 17-22 2016). http://www.dagstuhl.de/16162

[2] Philippe Kruchten, Robert L. Nord, Ipek Ozkaya: *Technical Debt: From Metaphor to Theory and Practice.* IEEE Software 29(6): 18-21 (2012)

[3] Philippe Kruchten, Robert L. Nord, Ipek Ozkaya, Davide Falessi: *Technical debt: towards a crisper definition report on the 4th international workshop on managing technical debt.* ACM SIGSOFT Software Engineering Notes 38(5): 51-54 (2013)

[4] *Managing Technical Debt in Software Engineering* Dagstuhl Reports, Vol. 6, Issue 4, April 17-22, 2016.http://www.dagstuhl.de/16162

[5] Antonio Martini and Jan Bosch, The Magnificent Seven: *Towards a Systematic Estimation of the Technical Debt Interest,* International Workshop on Managing Technical Debt (Cologne, Germany, May 22 2017).

[6] Antonio Martini, Simon Vajda, Mohamed Abdelrazek, Allan Jones, Rajesh Vasa, John Grundy, and Jan Bosch *Technical Debt Interest Assessment: from Issues to Project*, International Workshop on Managing Technical Debt (Cologne, Germany, May 22 2017).

[7] Jesse Yli-Huumo and Kari Smolander: *Changes and challenges of technical debt and its management during ongoing digital transformation,* International Workshop on Managing Technical Debt (Cologne, Germany, May 22 2017).

[8] Theodoros Amanatidis, Alexandros Chatzigeorgiou, Apostolos Ampatzoglou, and Ioannis Stamelos: *Who is Producing More Technical Debt? A Personalized Assessment of TD Principal,* International Workshop on Managing Technical Debt (Cologne, Germany, May 22 2017).

[9] Terese Besker, Antonio Martini, Jan Bosch, and Matthias Tichy: *An investigation of Technical Debt in Automatic Production Systems,* International Workshop on Managing Technical Debt (Cologne, Germany, May 22 2017).

[10] Thorsten Haendler, Stefan Sobernig, and Mark Strembeck: *Towards Triaging Code-Smell Candidates via Scenarios and Method-Call Dependencies*, International Workshop on Managing Technical Debt (Cologne, Germany, May 22 2017).

[11] Natthawute Sae-Lim, Shinpei Hayashi, and Motoshi Saeki: *Revisiting Context-Based Code Smells Prioritization: On Supporting Referred Context,* International Workshop on Managing Technical Debt (Cologne, Germany, May 22 2017).

[12] Sofia Charalampidou, Apostolos Ampatzoglou, Alexandros Chatzigeorgiou, and Paris Avgeriou: *Assessing Code Smell Interest Probability: A Case Study,* International Workshop on Managing Technical Debt (Cologne, Germany, May 22 2017).

[13] Philippe Kruchten: *Software architecture and agility: What colour is your backlog?*, Agile New England, July 7 2011.

[14] Eric Ries: *The Lean Startup: How Constant Innovation Creates Radically Successful Businesses*, Oktober 2011, Portfolio Penguin

[15] Kent Beck: *Extreme Programming Explained: Embrace Change, 2nd Edition*, Addison-Wesley