Report on the First International Workshop on Technical Debt Analytics (TDA 2016)

Aiko Yamashita
CWI, the Netherlands & HiOA, Norway
aiko.yamashita@cwi.nl

Leon Moonen
Simula Research Laboratory
Norway
leon.moonen@computer.org

Tom Mens
University of Mons
Belgium
tom.mens@umons.ac.be

Amjed Tahir
Massey University
New Zealand
a.tahir@massey.ac.nz

***Abstract*—This report outlines the motivation and goals of the First International Workshop on *Technical Debt Analytics* (TDA 2016), presents the workshop programme, introduces the work accepted for presentation, and summarizes the major results and themes that emerged from the discussion and activities undertaken during the workshop.**

I. Introduction

Technical debt (TD) is a metaphor reflecting technical compromises that can yield short-term benefit but may hurt the long-term health of a software system. This metaphor has been initially concerned with software implementation (i.e., code smells), but it has been extended to software design and architecture (i.e., anti-patterns and architectural smells) as well as documentation, requirements, and testing [1].

A systematic literature review by Li et al. [2] indicates that the term "debt" has been used in different ways by different software communities, leading to ambiguous interpretations of the term. They found that code-related TD (i.e., code smells) and its detection and resolution have gained the most attention whilst there is a need for more empirical studies with high-quality evidence on the whole Technical Debt Management (TDM) process and on the application of specific TDM approaches in industrial settings. The lack of empirically rooted evidence makes it difficult for organizations to align business value with the intrinsic quality of the software product itself. Zazworka et al. [3] argue that, in many projects, the cost and benefit of software refactoring (an approach to repaying TD) cannot be easily quantified and estimated. Consequently, it is still an open challenge to translate TD into economic consequences, making it difficult for development teams to make a strong case to the business side to them to invest in fixing technical shortcuts.

One of the major challenges rooted in the aforementioned "ambiguity" mentioned by Li et al. [2] is the lack of an underlying theory and models to aid TD identification and measurement. Seaman et al. [4] argue that a comprehensive TD theory should be developed to formalize the relationship between the cost and benefit of the TD concept, and subsequently practical TDM approaches should be developed and validated to exploit the TD theory in management decision making.

II. About TDA 2016

TDA 2016 was held in New Zealand on 6 December 2016, in conjunction with the 23rd Asia-Pacific Software Engineering Conference (APSEC 2016). The goal of TDA 2016 was to offer a specialised arena in TD to discuss about:

1) Calibrating technical debt and technical wealth related terminologies and concepts that are used indistinctly and interchangeably in software engineering literature.
2) Comparing, integrating, compiling and even reconciling empirical work on the effects of technical debt/technical wealth from economic and organisational perspectives.

To reach these goals, the workshop gathered practitioners and researchers working in the field of TD, to share experiences, concur on terminologies and evaluation guidelines, and to build a common research agenda for the community.

TDA 2016 built further upon results proposed during the Dagstuhl 16162 seminar on Managing Technical Debt in Software Engineering (April 2016), and discussed during the eighth international workshop on Managing Technical Debt (MTD), held in October 2016 in conjunction with ICSME.

III. Workshop Programme

The morning session of the workshop started with an introduction by the organisers, immediately followed by an invited keynote presentation entitled "*Towards quantifying technical debt*" by Ewan Tempero, Associate Professor in the Department of Computer Science at The University of Auckland, New Zealand. He discussed the current status of measuring TD and presented ideas as to what the TD community needs to do to develop the necessary tools to properly manage TD, and more specifically to quantify TD.

This keynote talk was followed by a short lightning talk by Jim Buchan on the relation between technical debt and legacy software. The extended abstract of his talk is reprinted, with permission, in section IV-C of this report.

The remainder of the morning session was filled with presentations of the accepted peer-reviewed contributions for TDA 2016. These are summarised in section IV-B.

The afternoon was devoted to a moderated discussion around the workshop goals, initiated by a brief summary by Clemente Izurieta (Assistant Professor at Montana State University) who presented the technical debt roadmap, future research perspectives and open research challenges discussed during the Dagstuhl seminar on Managing Technical Debt [5].

This presentation was followed by an interactive working session using discussion techniques such as card sorting and fishbowl panels. These sessions aimed at building a common understanding of the challenges, future directions for potential solutions and establishing a common research agenda.

IV. Workshop Contributions

A. Keynote address by Ewan Tempero: Towards quantifying technical debt

Summary: Technical debt (TD) is a metaphor that comes from the financial world, however it breaks down almost immediately. In the financial world when considering taking on debt, we can use a financial planner to determine such things as what our regular payments need to be and what the total cost of the loan will be. In software development, those making a decision that creates TD have no idea how much debt they are taking on, and often do not even realise when they are taking on some debt. For the metaphor to be useful, we must develop the means to quantify TD, in particular to be able to do so before we take on TD. In this talk I will discuss the current status of measuring TD and present some ideas as to what we have to do to develop the necessary tools to properly manage TD, specifically what we need to do to quantify TD.

Bio: *Ewan Tempero* is an Associate Professor in the Department of Computer Science at The University of Auckland, New Zealand. He graduated from the University of Otago, New Zealand, with a B.Sc., (Honours) in Mathematics in 1983 and received his Ph.D. in Computer Science from the University of Washington, USA, in 1990. He has published over 170 papers in journals and internationally-refereed conferences, mainly in the areas of software reuse, software tools, and software metrics. His current research is developing metrics for measuring the quality of software designs. He is the developer and maintainer of the Qualitas Corpus.

B. Accepted submissions for TDA 2016

The following papers were accepted to be presented during TDA 2016:

- Norihiro Yoshida. *When, why and for whom do practitioners detect technical debt? An experience report.*

  Based on his experience through industry-university collaboration, the author discusses when, why and for whom practitioners detect code clones, one of the most common code-level notions of technical debt.

- Yasutaka Kamei, Everton Maldonado, Emad Shihab and Naoyasu Ubayashi. *Using Analytics to Quantify Interest of Self-Admitted Technical Debt.*

  In this paper, the authors determined ways to measure the 'interest' on the debt and used these measures to see how much of the technical debt incurs positive interest, i.e., debt that indeed costs more to pay off in the future. To measure interest, they used the LOC and Fan-In code metrics, and carried out a case study on the Apache JMeter project.

- Solomon Mensah, Jacky Keung, Michael Franklin Bosu and Kwabena Ebo Bennin. *Rework Effort Estimation of Self-admitted Technical Debt information.*

  Programmers unintentionally leave incomplete, temporary workarounds and buggy codes that require rework. This phenomenon in software development is referred to as Self-admitted Technical Debt (SATD). The authors report on an exploratory study using a text mining approach to extract SATD from developers source code comments and implemented an effort metric to estimate the rework effort that might be needed to resolve the SATD problem. The study confirms the results of a prior study that found design debt to be the most predominant class of SATD. This technique could support managerial decisions on whether to handle SATD as part of on-going project development or defer it to the maintenance phase.

- Aabha Choudhary and Paramvir Singh. *Minimizing Refactoring Effort through Prioritization of Classes based on Historical, Architectural and Code Smell Information.*

  The authors present an approach for identifying and prioritizing object oriented software classes in need of refactoring by identifying the most change-prone as well as the most architecturally relevant classes, and by generating class ranks based on code smell information. Also, the approach provides to developers an estimation of maximum code smell correction (paying off maximum technical debt) with minimum refactoring effort.

- Johannes Holvitie, Sherlock Licorish, Antonio Martini and Ville Leppänen. *Co-Existence of the 'Technical Debt' and 'Software Legacy' Concepts.*

  Beyond strategic and accidental accumulation, technical debt may also occur due to delayed accumulation. In addition, technical debt and software legacy are concepts that share a lot of commonalities. Both concepts describe a state of software that is sub-optimal, and explain how this state can decrease an organization's development efficiency. The authors report on an initial examination of technical debt and software legacy similarities, and their somewhat challenging co-existence.

- Clemente Izurieta, Ipek Ozkaya, Carolyn Seaman, Philippe Kruchten, Robert Nord, Will Snipes, Paris Avgeriou. *Perspectives on Managing Technical Debt: A Transition Point and Roadmap from Dagstuhl*

  This paper summarizes the outcomes of a Dagstuhl Seminar where the current state of managing technical debt in software engineering was discussed. Participants reflected on the significant advances that the Managing Technical Debt (MTD) community has made since its inception in 2010; reached a consensus on a definition, called the Dagstuhl 16K

technical debt definition; and discussed avenues for future progress in the area. This paper offers a roadmap and a vision that describe the areas of research in TD where significant challenges remain.

C. Lightning talk by Jim Buchan: TD and legacy code

Many organisations have software that has evolved over many years and much of their focus is on enhancing and modifying this existing software product, some of which may be based on older technology. Although the older code may represent the company's core intellectual property, representing previous innovations, its age often introduces constraints and compromises on the continued evolution of the product. Over time, the proportion of the code judged as "legacy" grows, resulting in increased effort and uncertainty in expanding, testing and modifying the legacy code. At some point in time this may become untenable, with lost opportunity offered by new technologies, shortage of expertise in the legacy system, or unacceptable levels of bugs. Often this will trigger a full or partial re-write of the system to replace parts of the code.

The growth of the legacy code can be viewed as increasing technical debt (TD) in the sense that the software design has become sub-optimal over time and the interest in not paying back the legacy code debt may increase. The past decisions to incur debt may have a component that is deliberate, with the acceptance of compromises to new code, due to constraints of the legacy code. There is also a component of the TD that is not deliberate, with the emergence of new, unforeseeable technologies that offer new design and business opportunities.

I present a brief case study of an organization in New Zealand and its challenges and issues related to dealing with TD in the form of legacy code. The case organisation has a software product that grew very quickly in the size of the client base as well as the code base in the late 90s early 2000s. It was largely based on technology written in a 4GL language common at the time, with an integrated database and (limited) GUI input and output.

Over recent years the user interface and new modules were refreshed for a more modern look and feel, as well as to take advantage of the performance gains of new technologies. The changes were typically wrappers for the underlying 4GL code introducing new layers of processing, conversion, and presentation. There are over 1 million lines of code in the 4GL language and expertise in the 4GL language was becoming scarce. Extensions to the product were becoming increasingly difficult to develop and test with a high degree of uncertainty in dependencies and redundancies. After around 15 years of growth, the decision was made to initiate a project to port the existing product to a new technology stack, adding some new features at the same time.

In this presentation I will firstly establish a common vocabulary by exploring the meanings (and ambiguity) in the concepts of legacy code and TD and their relationship. Analysis of the case organization provides some grounded insights into some of the challenges and consequences of managing a large proportion of legacy code, as well as suggesting some recommendations for managing the legacy code debt. Based on the case study, as well as related research-based theory, I will address questions that include:

- When does code become "legacy"?
- How can the need to replace legacy code be identified?
- How can the economic value of a legacy code re-write be evaluated?
- What is the best way to approach the large re-write of a legacy code-base?

V. Themes that Emerged from the Workshop

The following research challenges and open issues emerged via discussions during the keynote, presentations, lightning talk and afternoon activities.

A. Heterogeneity of TD Definitions

One of the major obstacles to build a unified approach to quantify TD was found to be the lack of consensus on what constitutes TD. There are many different definitions of TD, leading to many different interpretations. For example, one way to define TD is do what you need to do to get a release on time. However, the interpretation of this definition is highly context-dependent and misses some important details on the potential effects of TD. It is widely believed that TD does not have a commonly agreed vocabulary (i.e., different terms can mean the same things). The SonarQube tool is a typical example of this issue, as it indicates TD for each rule violation. However, these violations are often project-dependent, making the tool displaying misleading or inaccurate results to determine TD.

B. Immature Measurement Theory in TD Studies

Another challenge for quantifying TD is the lack of understanding of general measurement theory and empirical assessment of software measurements [6]. For example, there is frequent misunderstanding or confusion between what metrics constitute in contrast to measurements.

C. Unclear Relation between TD and Legacy Systems

Another challenge raised was the difficulty of relating problems stemming from legacy systems with problems stemming from TD. What are the boundaries and the differences between both phenomena? It appears that managers in industry are unsure of what these boundaries are. During the workshop, a literature review was presented which constituted a mapping study to verify if the terms legacy and debt have been used together in previous studies.

D. Moderator and Contextual Factors

Another issue is the influence of moderator variables over TD. One of the case studies presented during the workshop indicated that the type of programming language used will highly influence the presence of code clones, thus one should be careful when assessing TD based on code cloning, for example by taking into consideration the language used, the framework available, etc.

E. Self-Admitted Technical Debt

Some advances presented and discussed during the workshop was the usage of text mining approach to identify instances of self-admitted technical debt (SATD), which can also lead to modelling and understanding better instances of TD, the context in which they were identified, as well as for estimations of reworks as result of SATD.

F. Technical Dept Research Community Agenda

During the workshop, the major outcomes from the Dagstuhl seminar on Managing Technical Debt were presented. That seminar tried to achieve the following goals:

- Identify the most pressing industry problems
- Identify the most promising research approaches
- Identify the "hard" research questions

A new definition of TD was proposed, focusing particularly on two quality aspects: maintainability and evolvability. The definition underscored the importance of the domain (ex. design or implementation issue) and the technical context (ex. degree of uncertainty, development and organisational context, time, causal chains).

Concerning a TD community agenda, it was deemed that the research and development of TD should lead to the following picture:

- More effort needs to be spent to develop a clear operational definition of minimum viable quality levels that can reconcile both technical and economic perspectives.
- There should be a clear way to translate developer concerns into manager concerns, which can be used as a basis for making decisions on investing on TW.
- TD would be incurred unintentionally most of the time.

VI. Results from the Card Sorting Activity

All participants were asked to post the TD terminologies that they were aware of on the whiteboard. We suggested three categories to classify terms: TD (Technical Debt), TW (Technical Wealth), and "Others" to mark relevant terms that did not directly contribute to TD or TW. For each single card it was discussed with the whole audience why this term should be included/excluded. We then mapped similar terms together, resulting in the following concept map of Figure 2.

Many of the participants suggested a wide variety of domains that they believe to be related to TD/TW, ranging from requirement gathering and analysis to deployment and maintenance. As shown in Figure 2, the majority of included TD terms are related to design TD. This has been referred to as anti-patterns, architectural smells, design flaws or poor architectural decisions. Some of the terms used by participants referred to coding-related TD issues such as code smells, ignoring standards, and poor programming practices.

Fig. 1. TDA participants during the card sorting activity

For TW, participants suggested that good design practices are the key to TW. Participants suggested that the use of design patterns and refactoring are considered valuable for achieving TW. A suggested example was the use of aspects (as in Aspect-Oriented Programming) and the implementation of the "separation of concerns" principle. Other suggestions that are likely to contribute to TW were the use of proper documentation, comments in the code, and applying coding standards. Many organisational factors are also considered valuable for TW. Examples are the awareness and acknowledgment of TD, the acknowledgment of additional maintenance cost and the risk of immature refactoring decisions.

VII. Results from the Panel Discussion

A panel discussion was moderated by Jim Buchan, featuring three panelists: Ewan Tempero, Clemente Izurieta and Yasutaka Kamei. The discussion focused on two out of seven topics selected by participants. The audience voted for the following questions: What is needed beyond more empirical studies? and What are the likely reasons studies appear to display contradictory results on the same smells (TD)?

Question 1: What is needed beyond more empirical studies?
Ewan asserted the need to think more about how to do studies and how to replicate some of these studies. He asserted that current studies in the SE community lack sufficient details to make replication of results possible. Clemente asserted that the goal of empirical studies needs to be clearly outlined, as well as their motivation. Yasutaka added that is important to have actionable results from these studies so that they can be used in industry.

Question 2: What are the likely reasons studies appear to display contradictory results on the same smells (TD)?
Ewan asserted that this is related to the first question and thus, the answer is the same. Good study design should show similar results. In order to do that, the experimental design should

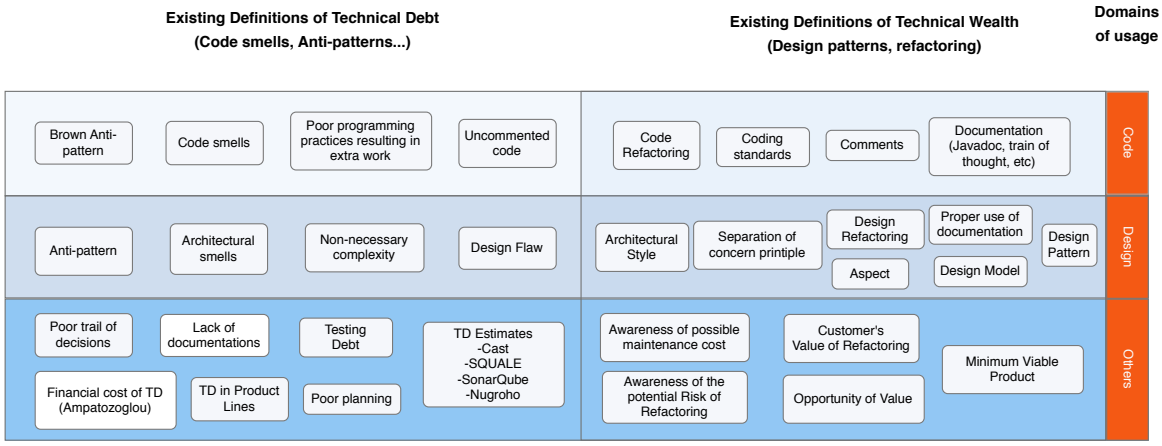

Fig. 2. Concept Map resulting from Sorting Activity

be clear enough. He remarked that in general, "...we usually have issues as software engineers to compare results of similar studies as we do not take other factors into consideration..." Clemente asserted that there are too many reasons, such as context, benchmark, no repository, poor methodology. He added that the question should be the opposite: how to show similar results? Yasutaka indicated that contradictory results are due to the studies depending on the context. If the context is different, then the results will look different.

Clemente added that there is a problem with students that are poorly prepared on how to conduct empirical studies, as well as to correctly produce and perform replication studies. This statement was confirmed by Ewan: students should learn more about research design/methods in order to be able to produce good empirical studies.

VIII. Workshop Summary

We summarise the main conclusions from the workshop in terms of identified challenges and subsequent strategies.

A. Challenges:

- We need a better classification of TD and standardise terminology to avoid confusion and quantify TD more accurately.
- Study replication is quite important in any empirical software engineering study, including TD. We should be able to replicate TD studies in order to be able to correctly compare results.
- More high quality empirical studies and evidence are needed, making replications possible and establishing an empirical basis and data science for TD.
- We need to better understand the interplay between model TD and implementation TD from methodological and instrumentation perspectives.
- Effective tooling needs to be developed to assist industry with assessing TD.

B. Strategy/Approach:

- Standardization efforts with members cross-cutting different TD domains are needed, via coordinated action events and/or a standardization task force. This should be initiated alongside cooperation with industrial practitioners.
- Incentives, frameworks and infrastructure need to be developed for facilitating the proliferation of Open Data, alongside support for describing the Methods used to Collect and Analyse the data. This can support and facilitate replication culture in SE research.
- Better preparation and culture for empirical replication must be supported by network actions and alongside industry-focused conferences (e.g., the XP conference).
- Better quantification (and tool support development) of TD can potentially be attained by:
  - Finding, curating, and providing accessibility to experimental artefacts
  - Reducing confounding factors by explicitly describing them and, when possible, controlling for them in empirical studies

IX. Acknowledgements

Our sincere gratitude goes to the PC members who helped in peer-reviewing the workshop submissions.

- Francesca Arcelli Fontana,University of Milano Bicocca
- Paris Avgeriou, University of Groningen
- Andrea Capiluppi, Brunel University
- Alexander Chatzigeorgiou, University of Macedonia
- Eleni Constantinou, University of Mons
- Steve Counsell, Brunel University
- Davide Falessi, California Polytechnic State University
- Yann-Gaël Guéhéneuc, École Polytechnique de Montréal
- Marouane Kessentini, University of MichiganDearborn
- Foutse Khomh, École Polytechnique de Montréal
- Ipek Ozkaya, Software Engineering Institute - Carnegie Mellon University
- Fabio Palomba, University of Salerno
- Gregorio Robles, Universidad Rey Juan Carlos
- Diomidis Spinellis, Athens University of Economics & Business
- Nikolaos Tsantalis, Concordia University
- Mel Ó Cinnide, University College Dublin

REFERENCES

[1] N. Brown et al. "Managing technical debt in software-reliant systems". In: *FSE/SDP workshop on Future of Software Engineering Research* (2010), p. 47.

[2] Z. Li, P. Avgeriou, and P. Liang. "A systematic mapping study on technical debt and its management". In: *J. Systems and Software* 101.11 (2015), pp. 193–220.

[3] N. Zazworka, C. Seaman, and F. Shull. "Prioritizing design debt investment opportunities". In: *2nd Workshop on Managing Technical Debt*. New York, New York, USA: ACM Press, 2011, p. 39.

[4] C. Seaman and Y. Guo. "Measuring and monitoring technical debt". In: *Advances in Computers* 82.25-46 (2011), p. 44.

[5] P. Avgeriou et al. "Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)". In: *Dagstuhl Reports* 6.4 (2016), pp. 110–138.

[6] N. Fenton and J. Bieman. *Software Metrics: A Rigorous and Practical Approach*. CRC Press, Nov. 2014.