Report from Dagstuhl Perspectives Workshop 24452

Reframing Technical Debt

Paris Avgeriou[*1], Ipek Ozkaya[*2], Heiko Koziolek[*3], Zadia Codabux[*4], and Neil Ernst[†5]

1 **University of Groningen, NL.** p.avgeriou@rug.nl
2 **Carnegie Mellon University – Pittsburgh, US.** ozkaya@sei.cmu.edu
3 **ABB Corporate Research – Mannheim, DE.** heiko.koziolek@de.abb.com
4 **University of Saskatchewan – Saskatoon, CA.** zadiacodabux@ieee.org
5 **University of Victoria, CA.** nernst@uvic.ca

Abstract

Technical Debt has undeniably become part of the everyday vocabulary of software engineers and has shaped the way both industry and academia think of tradeoffs made in software development, accounting for both value and cost: taking shortcuts to expedite software release (value) at the potential risk of higher cost for changes in the long term (cost). This trade-off is not to be taken lightly, as Technical Debt is considered by many software organizations to be the "silent killer" of software projects. To avoid being caught off guard, the industry is increasingly incorporating Technical Debt Management practices into their development processes. The research community has also produced a substantial body of knowledge on the topic of Technical Debt.

However, the industry requires more capable software tools that can manage both legacy and AI-generated code and specifically target Technical Debt. In addition, the industry needs practices and techniques that better support understanding the fundamental tradeoff: the value from incurring Technical Debt against Technical Debt's long-term costs. Progress on understanding this tradeoff is hindered due to a lack of high-quality datasets, as well as the comparatively small research effort focused on human and social aspects of Technical Debt.

Despite significant early progress in developing a common understanding of the concept of Technical Debt and code-related aspects, the research community and the software-intensive industry need to re-engage. Focusing on how Technical Debt Management is currently practiced and where to focus future research efforts was the goal of this Dagstuhl Perspectives Workshop 24452. The workshop brought together researchers, software tool vendors, and software practitioners to address open challenges and reframe the field of Technical Debt with a concrete and actionable manifesto.

The Dagstuhl Report documents the goals, format, and several discussions held during the workshop. The report also includes some of the outputs of the workshop, as well as the abstracts of the talks given by the participants. A key output of the workshop is a report that summarized a manifesto including values, beliefs, and principles of managing technical debt, titled *Reframing Technical Debt* and released as a separate document in the Dagstuhl Manifestos series.

Seminar November 3–8, 2024 – https://www.dagstuhl.de/24452

2012 ACM Subject Classification Software and its engineering → Software creation and management

Keywords and phrases Technical Debt, Software Maintenance and Evolution, Software Architecture, Software Economics, Software Quality, Socio-Technical Aspects of Software Development, AI-Augmented Software Development

Digital Object Identifier 10.4230/DagRep.14.11.16

[*] Editor / Organizer
[†] Editorial Assistant / Collector

1 Executive Summary

Paris Avgeriou (University of Groningen, NL)
Ipek Ozkaya (Carnegie Mellon University – Pittsburgh, US)
Heiko Koziolek (ABB Corporate Research – Mannheim, DE)
Zadia Codabux (University of Saskatchewan – Saskatoon, CA)
Neil Ernst (University of Victoria, CA)

Technical Debt is considered the "silent killer" of software projects. In a recent survey, software developers claimed they spend 13.5 hours weekly on Technical Debt, one-third of their working time[1]. The industry now has widespread consensus that managing Technical Debt should be treated as a core software engineering practice. Companies are increasingly incorporating Technical Debt management practices into their development processes [2], [6]. Many software quality tools now incorporate features to help software engineering teams visualize and triage quality issues within their code bases to assist Technical Debt management.

This advancement in industrial practice is paired with very vibrant research. The research community has produced a substantial body of knowledge on the topic of Technical Debt, especially in investigating the problem and its manifestation, understanding its urgency and impact, and proposing solutions to address it. This research is often performed in collaboration with industry, indicating its practical relevance but also the aligned interests of researchers and practitioners. The maturity of research in Technical Debt is evidenced by the increasing number of publications and systematic literature reviews since the first research paper was published in 2010 [1]. A recent tertiary study reviewing secondary studies in managing technical debt reported 532 unique research studies [3].

Despite the significant progress made in industry practices, commercial tools, and the research on the body of knowledge on Technical Debt Management, there are five critical gaps.

Lack of tools. While there is widespread use of static analysis tools to detect code quality issues, not all of them indicate Technical Debt, and there is significant confusion around their use. For example, several commonly used industry tools report different results for simple measures such as size, complexity, file cycles, and package cycles [4]. In addition, the software engineering community is unclear on what a Technical Debt management tool is.

Difficulty in understanding the impact of AI on Technical Debt. There is tremendous momentum around developing software with AI-based tools whose implications on new forms of Technical Debt are unclear. Consequently, while the industry is driving the simplification of code generation with AI-augmented tools to develop vast new quantities of code and develop it fast, it also risks incurring large volumes of unanticipated Technical Debt. Furthermore, the reality of industrial software engineering is dealing with a significant amount of legacy software. In these heritage systems, accumulated Technical Debt exists, but teams often lack resources to address the underlying issues. A large portion of that Technical Debt was introduced many years prior; it remains hidden and undocumented and involves design or architecture issues that cannot be mined by analyzing source code.

[1] https://stripe.com/files/reports/the-developer-coefficient.pdf

Lack of theory on Technical Debt economics. The bottom line of Technical Debt management is the Return On Investment (ROI) for the business and value, as difficult as it is to quantify the financial bottom line reliably. Existing research, while aimed to address complicated concepts such as the principal and interest of Technical Debt, fails to recognize that a new software economics approach is needed to concretely capture the cost of ownership and value aspects of Technical Debt.

Not focusing on architecture. Industry challenges consistently emphasize the difficulties around managing architecture-level Technical Debt where the tradeoffs are more implicit and complex. However, we observe the proverbial "lamp post" research, as researchers tend to focus on the easy part (code) rather than the more valuable but also more challenging part (architecture).

Narrow scope and data. There is not sufficient emphasis on the social side of Technical Debt Management. This is reflected in the limited research datasets: they only contain code artifacts and are not validated with human subjects. In addition, data sets that allow understanding of the socio-technical aspects of Technical Debt are lacking.

This Dagstuhl Perspectives Workshop brought together researchers, tool vendors, and software practitioners to analyze these gaps and reframe the field of technical debt, focusing on the following key challenge areas and guiding research questions:

- *Technical Debt as value-creation*: How can the value of Technical Debt be managed in an informed and conscious manner to meet business goals while still avoiding prohibitively high-interest payments in the future?
- *Elevating the role of architecture*: How can software architecture considerations be integrated into Technical Debt Management to elevate it beyond code-level issues?
- *Next-generation tooling*: How can new and existing tools better focus on Technical Debt Management, including taking advantage of AI-based capabilities where appropriate?
- *New perspectives on data collection*: How can forms of data beyond code analysis, such as architecture, documentation artifacts, and business artifacts, inform Technical Debt Management?
- *Socio-technical aspects*: Given that Technical Debt is often a matter of how individuals and teams operate, how can social aspects be integrated into Technical Debt Management?

Workshop Format

Before the workshop, a key reading list was distributed to the attendees to familiarize them with the most recent advances in Technical Debt Management. The workshop itself featured four elements:

- *Plenary Sessions* which included sharing organizational details, lightning talks on the five key challenges by all participants, reporting from the break-out groups, brainstorming on the vision, and planning ahead.
- *Break-out Sessions* which included brainstorming, discussions, and write-up sessions of the state of the art, as well as the values, beliefs, and principles of the manifesto, and finally, the roadmap towards realizing the manifesto. Each session had five break-out groups that corresponded to the aforementioned five key challenge areas.
- *Open Spaces* which allowed participants to propose and subsequently discuss additional related topics in a group setting.

- *Interactive and Inclusive Sessions* following some of the Liberating Structures[2]. These included activities such as "Impromptu Networking" to mine important challenges, a "User Experience Fishbowl" to share successes and concerns in both industry and research, a "Chaos Cocktail Party" to prioritize intermediate results, and a game to strengthen team bonding.

The participants in this workshop were carefully selected to obtain a balanced representation of the three main stakeholder groups in software engineering: (1) researchers whose work focuses on Technical Debt, (2) practitioners regularly dealing with Technical Debt, and (3) tool vendors who develop tools to address different aspects of Technical Debt Management. A list of participants is provided at the end of this report.

During the workshop, the lightning talks allowed the participants to get to know each other's backgrounds, work, and goals. The break-out sessions made up the majority of the workshop agenda and were the most intense sessions, producing hundreds of post-it notes, which were grouped into themes and subsequently transcribed to inform the resulting manifesto document. Each day, the break-out groups were slightly re-shuffled, with one or two new members joining each group to offer a fresh point of view. The outcomes of each break-out group were peer-reviewed by another group to offer feedback from a different perspective.

Various Open Spaces were proposed by workshop participants that covered a number of topics that complemented the five workshop topics: one open space dealt with "green" Technical Debt focusing on sustainability, another with different facets of Technical Debt (e.g., people, processes, business), and a third discussed how to create purpose-built Technical Debt benchmarks in a way they can be commonly accepted. The interactive sessions using the Liberating Structures were effective in energizing the participants after lunch and allowing a more engaging format to lead the discussions. As the workshop progressed, it became clear that the participants understood each other's perspectives better and became more efficient in their brainstorming and in-depth discussions.

Identifying Technical Debt Challenges

Following the lightning talks, the workshop agenda included an *impromptu networking* session. This activity aimed to provide participants an opportunity to interact with each other through an engaging activity and to elicit the challenges that the group collectively perceived as important.

Participants were asked to answer the following question on small cards: *What is the top priority challenge the Technical Debt community should make progress on to improve its management by software development teams?* The Technical Debt community here refers to researchers, tool vendors, and practitioners who are actively working on Technical Debt and its management. The participants then passed around their cards to read as many answers as possible. There were three impromptu pauses during this reading session, where the participants paired up and rated the challenge statements. This activity at the end of three rounds resulted in all cards having a rating by three different pairs. The top ten challenges the participants elicited are the following (in decreasing rating):

[2] https://www.liberatingstructures.com/

- Integrating tooling into existing development ecosystems in a non- or less-disruptive fashion
- Providing practical guidance to software development teams about when to accept Technical Debt
- Expressing the importance of Technical Debt to users, leaders, funding groups
- Helping C-level executives understand the strategic value of proper Technical Debt Management
- Developing techniques to prioritize technical debt so that the issues with the highest ROI can be addressed first to support efficiency, motivate developers/management to actually address Technical Debt, and avoid waste.
- Convincing business stakeholders to invest in Technical Debt repayment (not all Technical Debt has to be repaid) or not, including explaining the benefits in terms of money, but also risk management
- Extracting the context that underpins a team's Technical Debt, such as process, Technical Debt culture, and domain
- Easily determining the impact of Technical Debt reduction measures (e.g., regarding productivity gains, cost reduction, compliance mistakes) to support decision-making for product owners
- Developing tools that capture a commonly agreed Technical Debt viewpoint aligned with the company culture
- Transitioning from low-level code issues to architecture-level technical debt (architecture is defined as "expensive to change")

Industry/Academia Perspectives Session

A user experience "fishbowl" session on the first day highlighted several key insights about measuring and managing Technical Debt, capturing the different perspectives in industry and academia. While metrics were deemed often incomplete, subjective, or difficult to act upon, participants noted that the context around Technical Debt, for example, expressed in source code comments (also known as *self-admitted debt* [5]) could be utilized more in the future. Measuring Technical Debt and creating feedback loops can provide valuable data at varying levels, from developers to C-level executives. However, blindly optimizing specific metrics can lead to unintended consequences, such as wasted effort or undue pressure on teams. Empirical research today tends to focus heavily on source code analysis, missing the broader development and system context, which makes capturing a holistic view of Technical Debt challenging.

Looking forward, the fish bowl session underscored the importance of fostering collaboration between industry and academia as well as including interdisciplinary perspectives. Industry participants highlighted the relevance of team roles (e.g., having a dedicated role in overseeing Technical Debt Management), team rules (e.g., having design guidelines), and management dynamics (e.g., educating non-technical management on Technical Debt), while academic perspectives leaned toward analyzing easily accessible artifacts, such as source code. Both groups agreed on the need to address the human aspects of Technical Debt and the variability of metrics, which some view as a problem and others as an opportunity.

To improve Technical Debt Management, industry participants recommended appointing a "Technical Debt champion" within software development teams/organizations to educate and advocate for best practices, especially towards management. They also suggested adapting research methods from other disciplines, such as social sciences, creating more purpose-built Technical Debt Management tools that seamlessly integrate into developer and decision-maker workflows, and demonstrating how Technical Debt impacts innovation to secure buy-in from

upper management. These approaches aim to create more actionable strategies for addressing Technical Debt while maintaining alignment with organizational goals. The fishbowl session was thus instrumental in eliciting some of the participant's core values, beliefs, and principles regarding Technical Debt, which is at the heart of the manifesto.

Open Space on the Multi-Dimensionality of Technical Debt

Technical Debt emerges from decisions or other phenomena (e.g., deprecation of a library) throughout the software development cycle, jeopardizing software maintenance and evolution and affecting most development workflows. However, managing Technical Debt takes significant effort and time. One of the main reasons is that different context variables and factors (peopleware, processes, products, business, etc.) influence how Technical Debt is managed or even perceived. It certainly goes far beyond source code, and there is no single metric that can accurately capture the concept. This Open Space discussed ways to measure Technical Debt, taking those multiple perspectives into account, as well as evidence-based metrics required to support decision-making.

The participants agreed that collecting both objective and subjective data is necessary, as well as a way to aggregate these two types. This aggregation should take stakeholders' concerns into account (what stakeholders want to know to make decisions). Quantitative measures can be contextualized by qualitative ones. The combination of measures must represent the context and match the organization; context is key.

Representation of Technical Debt also requires multiple metrics; a single number is hard to understand and interpret and may hide or obscure the complexities underlying Technical Debt. In addition, graphical representations are useful but must be followed by an explanation (recommendation) to provide value to stakeholders (CEO, Project Owner, developers, and so on).

Beyond metrics, Technical Debt items must include other contextual information. How exactly does it hinder development? Why was it incurred in the first place? This kind of context should drive data collection. Sometimes, specific Technical Debt items need different metrics and contextual information as they affect development differently.

There is still much tacit knowledge involved in managing technical debt, especially in identifying, measuring, prioritizing, and repaying it. Therefore, experienced software developers have a relevant role in mentoring their more junior colleagues in understanding the software structure and behavior, as well as past design decisions. Traceability between software artifacts (e.g., between source code and documentation) can facilitate making this tacit knowledge explicit. However, in general, software professionals must be prepared to spend time understanding the context behind technical debt items.

Based on the aforementioned issues, the Open Space participants identified several research directions:

- How to identify meaningful objective metrics that should be automatically collected?
- How to measure and manage developer "turnover" (e.g., the departure of developers with knowledge of legacy code)?
- How to capture current developers' perspectives about Technical Debt?
- How to record the design decisions that incur Technical Debt?
- How to aggregate different unit metrics?
- What is the interplay between Indicators of Technical Debt and Metrics of Technical Debt?
- How to incorporate a human-in-the-loop within the Technical Debt management process?
- How to adequately capture the balance between Technical Debt repayment and the development of new features, especially regarding quality, risks, and costs?

Manifesto

The core outcome of this workshop is a report published in the Dagstuhl Manifestos series[3], titled *Reframing Technical Debt*. This manifesto report summarizes the current landscape of Technical Debt methods and tools and explains where current industrial practice is struggling, where current research has made progress, and where it falls short. This motivates the actual manifesto, which condenses the outcomes of the workshop into a series of **values, beliefs, and principles**.

The manifesto reports subsequently explains the stated values, beliefs, and principles in detail. Finally, it outlines a roadmap with concrete milestones and a call to action addressed at researchers, software practitioners, tool vendors, Technical Debt advocates, policymakers and other government agencies, leading technology companies, funding agencies, industry leaders, and industry associations. The manifesto is signed by the workshop participants.

Concept Map

During the lightning talks and plenary discussions, the organizers captured key concepts discussed and how they related to each other as a concept map. This concept map (Figure 1), though not exhaustive, offers a perspective on the multidisciplinary nature of Technical Debt, its management, and its essential role in software engineering.

The rest of this report includes abstracts from workshop attendees summarizing their relevant work, which informed the discussions during the workshop.

References

1 Nanette Brown, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert Nord, Ipek Ozkaya, Raghvinder Sangwan, Carolyn Seaman, Kevin Sullivan, and Nico Zazworka. Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, page 47–52, New York, NY, USA, 2010. Association for Computing Machinery.

2 Ciera Jaspan and Collin Green. Defining, measuring, and managing technical debt. *IEEE Softw.*, 40(3):15–19, May 2023.

3 Helvio Jeronimo Junior and Guilherme Horta Travassos. Consolidating a common perspective on technical debt and its management through a tertiary study. *Information and Software Technology*, 149:106964, 2022.

4 Jason Lefever, Yuanfang Cai, Humberto Cervantes, Rick Kazman, and Hongzhou Fang. On the lack of consensus among technical debt detection tools. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 121–130, 2021.

5 Aniket Potdar and Emad Shihab. An exploratory study on self-admitted technical debt. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 91–100. IEEE, 2014.

6 Wolfgang Trumler and Frances Paulisch. How "specification by example" and test-driven development help to avoid technial debt. In *2016 IEEE 8th International Workshop on Managing Technical Debt (MTD)*, pages 1–8, 2016.

[3] https://drops.dagstuhl.de/entities/journal/DagMan

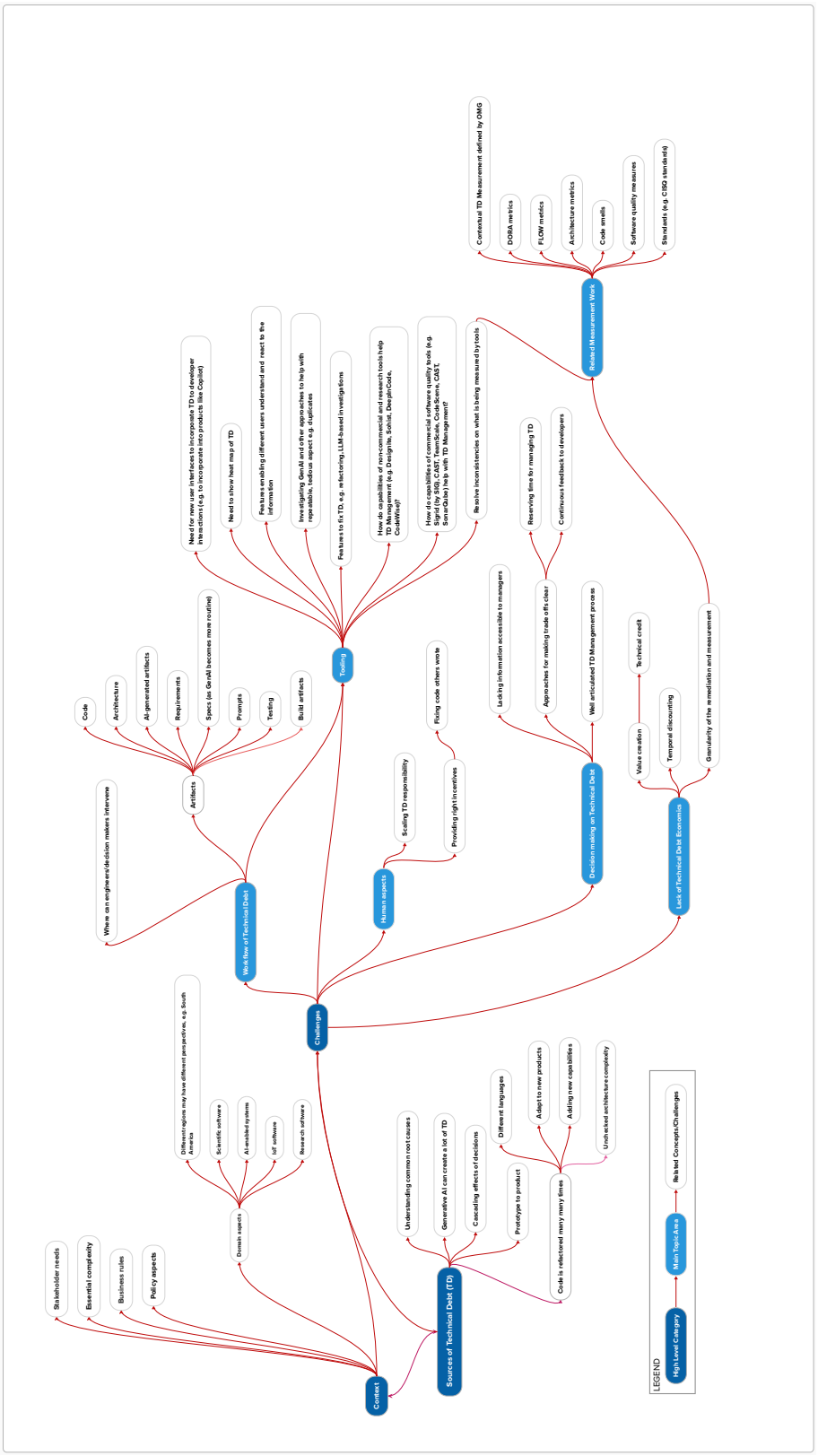

Figure 1 Technical Debt Concept Map.

2 Table of Contents

3 Overview of Talks

3.1 Explainable AI and Two-Way Learning for Technical Debt Management

Apostolos Ampatzoglou (University of Macedonia – Thessaloniki, GR)

Joint work of Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Nikolaos Mittas, Dimitrios Tsoukalas, Elisavet-Persefoni Kanidou, Nikolaos Nikolaidis, Theodoros Maikantis, Elvira-Maria Arvanitou
Main reference Dimitrios Tsoukalas, Nikolaos Mittas, Elvira-Maria Arvanitou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Dionysios D. Kehagias: "Local and Global Explainability for Technical Debt Identification", IEEE Trans. Software Eng., Vol. 50(8), pp. 2110–2123, 2024.
URL https://doi.org/10.1109/TSE.2024.3422427

In recent years, we have witnessed a significant increase in research focusing on how Machine Learning (ML) techniques can be used for software quality assessment and Technical Debt Identification. However, the derived methodologies and tools lack transparency due to the black-box nature of the employed machine learning models, leading to decreased trust in their results. In addition, refactoring is the most prominent way to repay Technical Debt in the software development industry. In the literature, numerous tools provide refactoring suggestions based on well-established approaches. However, two of the main shortcomings of these approaches are: (a) that these tools usually provide many refactoring suggestions that the developer cannot parse, and (b) in several cases, the software engineers disagree with some suggestions. These problems decrease the trust of developers in the refactoring tools/approach, eventually leading engineers to drop out of using refactoring support. The lack of human trust in AI-based decision support systems has (among others) led to the rise of the ethical AI movement, calling for transparency and trustworthiness of AI systems. From our perspective, these challenges open up a new research direction in the field of Technical Debt Management that can introduce approaches relying on Explainable AI and Two-Way Learning methodologies for making the cooperation of software engineers and AI systems more efficient and acceptable for humans, putting them more actively in-the-loop and at the center of the software quality assurance process.

3.2 Tools for Practical Technical Debt Management

Lodewijk Bergmans (Software Improvement Group – Amsterdam, NL)

URL https://www.softwareimprovementgroup.com/sigrid-software-excellence-platform/

Software Improvement Group provides tools and consulting to software development organizations for -among others- measuring quality (including maintainability and architecture sustainability) and Technical Debt in their software systems. These services are applied incidentally as assessment and advice services, periodical reporting, and continuous monitoring.

We propose the following principles to make such services trustworthy and practical for practitioners:

- Objective and fact-based: based on (static) analysis of code and other artifacts
- Adoption of international standards and use of well-defined quality models provide transparency

- Automated and repeatable measurements to support continuous, "real-time" monitoring without manual effort
- Following a risk-based approach that aggregates fine-grained findings based on the risk categories of those findings to ensure that the conclusions are dominated by the presence or absence of higher-risk issues
- Benchmarking ensures that results provide relevant ratings and the ability to set feasible objectives, as well as fair comparison across teams, systems, and organizations

Here are some examples of models and tools that are in use by our customers for each of the Technical Debt topics identified for the Dagstuhl:

- Value-creation: a quality model for code maintainability, an architecture quality model for architecture sustainability and evolvability, and a qualitative model for calculating the Technical Debt principal based on measurements on the code base
- Tooling: tooling that supports continuous monitoring, integration in CI/CD pipelines, dashboards of Technical Debt and its trends, and visualization of architectures
- Architecture: an architecture quality model that quantifies the sustainability and evolvability of an architecture and a model for reverse engineering the realized architecture
- Data collection: Technical Debt data is derived from source code snapshots, (deltas of) quality ratings, SCM-based info such as commit information
- Socio-technical aspects: for example, the latter information is used to identify and report socio-technical aspects about team organization (vs. the architecture) and team interactions

Applying these models in practice has shown their ability to provide useful insights, which can help development organizations manage their Technical Debt better.

3.3 Maintaining Stamina and Motivation in Technical Debt Management

Markus Borg (CodeScene – Malmö, SE)

Joint work of Markus Borg, Adam Tornhill
Main reference Markus Borg, Ilyana Pruvost, Enys Mones, Adam Tornhill: "Increasing, not Diminishing: Investigating the Returns of Highly Maintainable Code", in Proc. of the 7th ACM/IEEE International Conference on Technical Debt, TechDebt 2024, Lisbon, Portugal, April 14-15, 2024, pp. 21–30, ACM, 2024.
URL https://doi.org/10.1145/3644384.3644471

Technical Debt Management is essential but often sidelined over time, even in organizations initially committed to it. At CodeScene, we use the CodeHealth metric to identify and prioritize refactoring targets. CodeHealth is an aggregated file-level metric based on established code smells known to increase the cognitive load on developers. Empirical studies on proprietary projects have shown strong links between CodeHealth, defect density, and development velocity. Yet, maintaining momentum in CodeHealth improvement remains challenging for many organizations.

To address this, we are developing value models from anonymized customer data to tie Technical Debt remediation to business value. Additionally, we analyze code smell removal during pull requests to estimate the value of prevented CodeHealth declines. Our aim is to equip organizations with "what if" insights to support return on investment analyses for ongoing maintainability improvements, ultimately helping them sustain focus on code-level Technical Debt Management.

3.4 The Importance of Agreement in Technical Debt Identification

Alexander Chatzigeorgiou (University of Macedonia – Thessaloniki, GR)

Joint work of Alexander Chatzigeorgiou, Theodoros Amanatidis, Athanasia Moschou, Nikolaos Mittas, Apostolos Ampatzoglou, Lefteris Angelis
Main reference Theodoros Amanatidis, Nikolaos Mittas, Athanasia Moschou, Alexander Chatzigeorgiou, Apostolos Ampatzoglou, Lefteris Angelis: "Evaluating the agreement among technical debt measurement tools: building an empirical benchmark of technical debt liabilities", Empir. Softw. Eng., Vol. 25(5), pp. 4161–4204, 2020.
URL https://doi.org/10.1007/S10664-020-09869-W

The lack of a universal agreement on what constitutes Technical Debt and how it manifests in software artifacts – especially in source code – poses challenges for Research, Practice, and Education in Software Engineering: The absence of a commonly agreed-upon "ground truth" for Technical Debt instances threatens the validity of new approaches and empirical studies. Tools for Technical Debt Management used in practice also suffer from limited credibility, as different approaches produce widely varying Technical Debt estimates and recommend divergent mitigation actions. Furthermore, without a widely accepted benchmark, effectively training students and young professionals becomes challenging. However, aligning findings from multiple Technical Debt identification approaches/tools and validating them with expert feedback could establish trustworthy reference datasets. Such Technical Debt benchmarks would create valuable opportunities for Technical Debt Management, enabling Machine Learning models to be trained and adapted to domain-specific Technical Debt issues.

3.5 Contextualizing Technical Debt for Better Insights

Zadia Codabux (University of Saskatchewan – Saskatoon, CA)

While code smells and static analysis tools are widely used to identify Technical Debt, they often provide an incomplete picture. By primarily focusing on surface-level issues, such as poor code structure or style violations, we fail to capture the broader factors that contribute to Technical Debt. This limitation is more evident in non-traditional or scientific software domains, where performance trade-offs, domain-specific constraints, long-term sustainability considerations, and development priorities often diverge from conventional software engineering practices. To address these challenges, there is a need for a more context-driven approach integrating socio-technical considerations, domain-specific nuances, and a deeper understanding of trade-offs beyond what current tools can offer to better align Technical Debt Management practices with the constraints pertaining to diverse software ecosystems.

3.6 Who Owns Technical Debt? Building a Proactive Culture

Stefano Dalla Palma (Adyen – Amsterdam, NL)

Drawing from my experience training and mentoring developers, I've observed widespread enthusiasm for software quality principles, yet many struggle to apply these practices when it matters most. Junior developers are eager to contribute through feature development but often lack the experience to recognize or address Technical Debt. Meanwhile, senior developers, often preoccupied with management responsibilities, can disengage from maintaining code quality over time. Adding to this challenge, assigning Technical Debt Management to a dedicated team isn't scalable and can undermine individual responsibility, as other teams may expect them to shoulder the burden. This brings us to a larger question: Is Technical Debt merely a technical issue, or is it ultimately about fostering a culture of shared ownership? Moving forward, the key lies in continuously engaging technical and business stakeholders through channels such as dedicated training, reading groups, and other collaborative forums. By creating a knowledge-sharing community where everyone feels empowered to learn about and discuss Technical Debt openly, we can promote a culture of software quality best practices where addressing Technical Debt becomes a shared responsibility.

3.7 Feedback Loops are Key to Effective Software Quality Control (and Technical Debt Management)

Florian Deissenboeck (CQSE – München, DE)

URL https://teamscale.com/

Academic research offers a well-defined concept of Technical Debt, distinguishing it from other software quality issues. In contrast, industry practitioners use the term more broadly to encompass any quality defect. While research focuses on Technical Debt within source code, practitioners find that architectural shortcomings or outdated technology stacks (e.g., obsolete libraries) often pose costs comparable to code-related Technical Debt. Research highlights that accepting Technical Debt involves trade-offs, such as accelerating time-to-market. Yet, in practice, these decisions are frequently made unconsciously, with their adverse effects manifesting much later.

The distinction between academia's precise view and industry's broader interpretation of Technical Debt is less critical than one might think. The tools and processes required to tackle these software quality challenges are mainly identical. A key strategy for managing Technical Debt or general software quality is implementing a feedback loop. This loop incorporates a comprehensive software system analysis, examining not just the source code but also architecture, technology stacks, and other artifacts like build scripts, requirements specifications, and test cases. Insights from the development process, such as defect data or ticket resolution times, also feed into this loop.

To cope with today's large and complex systems, this analysis should be as automated as possible while allowing for necessary manual interventions for aspects that cannot (yet) be automated. The results of the analysis are translated into actionable tasks for software engineers. Trade-offs inherent in software development are best managed when the feedback process is moderated by a role empowered to make such decisions, like a product owner.

Over the past decade, CQSE has assisted various organizations in establishing these data-driven feedback loops, effectively managing Technical Debt and improving the quality of hundreds of software systems encompassing millions of lines of code. Our experience demonstrates that this approach is cost-effective and significantly enhances software quality without sacrificing the ability to make strategic trade-offs, such as temporarily accepting certain issues to expedite time-to-market.

3.8 Advancing Technical Debt Management Post-ISO/IEC 5055:2021

Philippe-Emmanuel Douziech (CAST – München, DE)

The adoption of the Object Management Group (OMG) Automated Source Code Quality Measures (ASCQM) standard as ISO/IEC 5055:2021 prompted the Consortium for IT Software Quality (CISQ) to revisit its Automated Technical Debt Measure (ATDM) to align with this new standard and integrate insights from recent experiences, particularly regarding the use of structural insights to prioritize Technical Debt better.

Despite these efforts, significant challenges persist with ATDM adoption. Technical Debt is still broadly perceived negatively, with improved identification of ATDM items often proving counterproductive by visibly increasing the "Technical Debt principal." The additional focus on architecture-level ATDM items introduced in ISO/IEC 5055:2021 has only amplified this effect.

The popularity of the Technical Debt metaphor, while helpful in raising awareness, often creates a false sense of comprehension regarding its effective management. The metaphor oversimplifies the reality of Technical Debt in software engineering, where unique approaches can be applied to reduce debt, especially when concentrated debt becomes unsustainable. However, current models are inadequate in capturing critical decisions, such as the choice to retire and replace debt-heavy components, instead of addressing each Technical Debt item individually. The lack of mature models for these strategies means they are largely absent from aggregated measurement results.

Although the Contextual Technical Debt Measure (CTDM) was introduced in the ATDM standard to account for project-specific contexts and encourage adoption by development teams, uptake remains slow. Companies often prioritize Technical Debt measures for inter-project and team comparison rather than using them to empower teams to manage their debt within their own contexts.

Even with tools that can automatically detect architectural nonconformities and generate "as-is" architectural blueprints, architecture-level Technical Debt is frequently overlooked.

The recent gathering in Dagstuhl, Germany, of representatives from research, industry, and software vendors provides a valuable opportunity to leverage collective experience, address these ongoing challenges, and chart a productive course for the future.

3.9 The Role of Domain Knowledge in Technical Debt

Neil Ernst (University of Victoria, CA)

Domain knowledge does not equal software expertise. Domain knowledge is about business rules, stakeholder needs, and essential complexity. Technical Debt means nothing in the abstract. Everyone has it, but no fixes are common. Rules and metrics are useless out of the box, useless at a portfolio level, and maybe useless at a project level; they need **context**. This is especially challenging when the domain knowledge is highly technical (e.g., radio astronomy or climate modeling). To resolve this, we can automatically extract context using, for example, machine learning assessment of key problems or longitudinal data on previous priorities. We can also use qualitative insights (e.g., Quality Attribute Workshops) or developer surveys. Longer term, we might try large language model recovery of relevant domain context, although how helpful this is, is unclear, such as labeling Technical Debt instances or assessing impacts. Incorporating domain knowledge will make Technical Debt Management more relevant.

3.10 Workflow-Driven Technical Debt Management

Daniel Feitosa (University of Groningen, NL)

Technical Debt remains a critical challenge in software development, appearing in all aspects of the development and operations workflow, including code, documentation, tests, deployment, logging, and monitoring. Symptoms of accumulating Technical Debt can be found in such artifacts, which are produced by either human actions or automated tools.

To effectively manage Technical Debt, integration into the existing team's workflow is essential, ensuring synergy rather than imposing an additional burden. This integration must capture the perspectives of different stakeholders, including developers, team leads, and product managers, to account for technical, operational, and business needs. By bridging practical challenges and research insights and grounding benchmarking in quantitative metrics and qualitative developer experiences, Technical Debt Management can evolve into a holistic process that considers multiple dimensions of software quality.

A critical approach involves combining evidence from various workflow artifacts – akin to a "sensorial fusion" – to understand the larger picture of Technical Debt. By correlating symptoms from tools and developers' perspectives, it is possible to improve identification, prioritization, and targeted intervention strategies.

Effective Technical Debt Management is not about eliminating debt entirely but about making informed trade-offs that respect the evolving business and development context. Shared ownership, human perception, and psychological safety are key to openly documenting Technical Debt, empowering teams to discuss and make context-aware decisions. A workflow-driven Technical Debt Management approach should align technical challenges and business objectives to pave the way for more sustainable and efficient software development processes.

3.11 Practical and Sustainable Approaches to Managing Technical Debt

Collin Green (Google – Mountain View, US)

Joint work of Collin Green, Ciera Jaspan, The Google Engineering Productivity Research Team
Main reference Ciera Jaspan, Collin Green: "Defining, Measuring, and Managing Technical Debt", IEEE Softw., Vol. 40(3), pp. 15–19, 2023.
URL https://doi.org/10.1109/MS.2023.3242137

The Google Engineering Productivity Research Team has been examining Technical Debt as a hindrance to developer productivity for several years. Our work on Technical Debt employs a variety of quantitative and qualitative methods but has focused heavily on survey research with quantitative and qualitative analysis of structured survey items and open-ended questions. Initially, we sought to better define what engineers mean when they talk about Technical Debt, iteratively refining how we ask about this topic in survey questions and how we measure and report it. Presently, we measure and report on ten types of Technical Debt that vary in their sources, implications, and severity. Our team has also collaborated with expert practitioners at Google on frameworks and processes for the effective management of Technical Debt. Our latest research has examined engineers' comments regarding Technical Debt in the context of the recent increased focus on achieving higher engineering velocity (and perhaps, accordingly, less caution about incurring Technical Debt). Our aim is to achieve a better understanding of how processes and structure can help engineering teams have a healthy, sustainable, and practical approach to incurring, managing, and paying back Technical Debt.

3.12 Correlating Technical Debt Metrics With Developer Perceptions

Ciera Jaspan (Google – Mountain View, US)

Joint work of Ciera Jaspan, Collin Green, The Engineering Productivity Research team at Google
Main reference Ciera Jaspan, Collin Green: "Defining, Measuring, and Managing Technical Debt", IEEE Softw., Vol. 40(3), pp. 15–19, 2023.
URL https://doi.org/10.1109/MS.2023.3242137

Our research team at Google takes a mixed-methods approach to understanding and measuring Technical Debt. Our surveys give us developer perceptions of the impact of 10 forms of Technical Debt and are regularly used for prioritization. However, both engineers and managers report the need to see a "heatmap" of where Technical Debt is in the code; subjective data is helpful for knowing there is a problem, but it doesn't provide the extent of impact. The objective data would be helpful for relative prioritization, discussions with stakeholders, and evaluating the impact of company-wide improvements to processes and tools that are intended to better fix or manage debt.

To evaluate potential objective metrics of Technical Debt, we attempted to correlate them with developer perceptions from the survey. We did this for three forms of Technical Debt but found no correlations with 117 metrics. It may be that these metrics are insufficient at their core, but it also may be that our research method was flawed as we had to make assumptions about whether a developer may have encountered each Technical Debt issue in their work. It also might be that Technical Debt, at its essence, is the difference between what the system is and what the developer believes the system "ought to be," and therefore,

this problem may be inherently intractable. More research needs to be done, and we call on others in the community to validate Technical Debt metrics against developer perceptions of the impact of Technical Debt on productivity.

3.13 Boeing Industry Perspective – Reframing Technical Debt

Ron Koontz (The Boeing Company – Mesa, US)

Joint work of Ron Koontz, Austin Vincent

Technical Debt is pervasive within all software; it exists across all development phases and grows unchecked when not deliberately managed. Accumulated Technical Debt is debilitating and an existential threat to long-life software systems. Proactive Technical Debt Management streamlines construction and integration of new software features and provides decision trade-off insight (potential Technical Debt prevention) across the software ecosystem.

Technical Debt measurement tools merged into the software development environment empower the software engineering experience while yielding higher quality, easier to change software products. When applied to reusable software, full-lifecycle Technical Debt Management leads to broader acceptance and adoption.

A practical Technical Debt Management implementation includes 3 fundamental components: 1) Technical Debt Primer, 2) Technical Debt Inventory Process and 3) Technical Debt Management System. The Technical Debt Primer captures organizationally consistent terminology, common understanding and complete descriptions for the Technical Debt Item Inventory and the Technical Debt Management System. The Technical Debt Inventory Process is an integrated software development activity employing a tool-agnostic analysis and decision framework with predefined and contextual attributes. Software developers identify/define, prioritize, and repay (closeout) Technical Debt Items within the backlog, concurrent with feature development and defect correction, as part of Agile workflows. The Technical Debt Management System utilizes automated (and limited manual) metrics capture, hierarchically integrated within dashboards, to inform software managers and leadership on Technical Debt concerns using data analytics.

Known Technical Debt Management Gaps remain:

1. There is no universal set of Technical Debt Management metrics
2. Technical Debt benchmarks are insufficient; standardization is needed to consistently measure and track Technical Debt
3. Code Technical Debt Management tools are in their infancy; tools to measure other forms of Technical Debt, e.g., architecture, are embryonic
4. A universal set of Technical Debt root causes should be defined to provide consistent reference for risk management and impact analysis.

The 2024 Dagstuhl experience has brought together practitioners, researchers and the tooling community. Together, we're poised to take the next step – to further refine, automate and validate the Technical Debt Management process.

3.14 Managing Technical Debt in Industrial Software Systems: Dedicated Architecture Technical Debt Management and Interactive LLM Agents.

Heiko Koziolek (ABB – Mannheim, DE)

URL https://www.abb.com

ABB, a major company in the industrial automation domain and develops software-intensive systems that include distributed systems, servers, real-time controllers, edge gateways, intelligent devices, cloud backends, and mobile applications. The value of ABB's products and systems is increasingly driven by software, but the presence of Technical Debt in this software can be problematic. This is because industrial automation applications often need to be maintained for extended periods. Despite their lower change velocities compared to consumer software, Technical Debt in these applications still incurs interest payments, requiring costly migrations and updates.

Past efforts to tackle Technical Debt have involved applying source code analysis tools and surveying developers. While these approaches have some benefits, such as making Technical Debt visible and facilitating communication, they are often incomplete and lack structured processes. A culture that recognizes the importance of managing Technical Debt is crucial, but it must also balance the risk of over-engineering with the need for reasonable, cost-effective efforts in industrial domains.

Future approaches to managing Technical Debt should include purpose-built tools that address architectural debt. These tools need to be capable of handling informal architecture sketches, incorporating external information about component suppliers, supporting technology decisions, and aligning with business goals. They should also suggest architecture best practices and provide proven solutions. Although current program analysis tools are helpful in identifying and visualizing code smells, there is a gap in dedicated, agile, low-overhead solutions for managing Technical Debt.

Customizable LLM-based agents offer a promising solution to support developers by identifying Technical Debt, framing existing issues, and brainstorming approaches to address them, such as re-designing parts of the code. These agents could help developers generate appropriate source code and foster a culture of continuous Technical Debt Management, integrating seamlessly into regular development practices.

3.15 Enhancing Developer Experience Through Systematic Technical Debt Management at SAP

Christof Momm (SAP – Dresden, DE)

Joint work of Christof Momm, Paige Perusset, Jochen Rode

Technical debt is an inevitable aspect of software development that impacts innovation, maintenance, and efficiency. For large companies like SAP, managing Technical Debt systematically can save costs and boost productivity, but this requires a cultural shift.

A shared understanding of the value and risks associated with Technical Debt among both business and technical stakeholders is essential. This can be fostered through straightforward, user-friendly processes for managing Technical Debt, supported by tools that minimize developer overhead while accommodating necessary complexity under the hood.

At SAP, we have taken/are planning several measures to improve the management of Technical Debt:

- Playbook: Creating and distributing a straightforward playbook for systematic Technical Debt Management, along with providing training sessions on how to prevent and handle Technical Debt
- Feedback: Regularly gathering developer feedback on Technical Debt based on our developer experience survey
- Technical Debt Budget: Elevating Technical Debt items to first-class citizens in portfolio and sprint planning, securing the necessary capacity and budget for effective management and resolution
- Proof of Benefits: Demonstrating the return on investment (ROI) from reducing Technical Debt, such as increased developer satisfaction, improved Technical Debt feedback, cost savings, and increased development velocity, by leveraging central DORA and FLOW metrics as well as user research
- Technical Debt Mitigation Techniques: Providing a "toolbox" for mitigating common types of Technical Debt more efficiently. This includes large-scale refactoring techniques to address migration debt and leveraging GenAI for validating and enhancing service and API documentation

Despite our advancements, challenges remain that require further investigation:

- Developing leading indicators/early warning signals for identifying critical Technical Debt and supporting decision-making (cost vs. value of Technical Debt)
- Providing Technical Debt Management tools that integrate seamlessly with existing development processes and tools and leverage all available data.

Addressing these key issues will require a joint effort from researchers, tool vendors, and practitioners.

3.16 United States Department of Defense Technical Debt Study

Brigid Petrie O'Hearn (Carnegie Mellon University – Arlington, US)

Main reference Ipek Ozkaya, Forrest Shull, Julie B. Cohen, and Brigid O'Hearn: "Report to the Congressional Defense Committees on National Defense Authorization Act (NDAA) for Fiscal Year 2022 Section 835 Independent Study on Technical Debt in Software-Intensive Systems," Carnegie Mellon University, Software Engineering Institute's Digital Library, Technical Report CMU/SEI-2023-TR-003, 2023
URL https://doi.org/10.1184/R1/24043392

This talk focused on the Report to the Congressional Defense Committees on National Defense Authorization Act (NDAA) for Fiscal Year 2022 Section 835 Independent Study on Technical Debt in Software-Intensive Systems. It emphasized unique United States Department of Defense (DoD) challenges but highlighted the socio-technical aspects of the findings and recommendations. DoD, having many legacy systems with significant Technical Debt and strict funding rules (colors of money), needs to share best practices and provide additional guidance to individuals and teams empower programs to incorporate Technical Debt Management into software development lifecycle activities as a core software engineering practice. The talk also noted that DoD should invest in/utilize Technical Debt research areas as well as improve tool support and data collection. Technical Debt Management must become a valued component of sustainable growth and product quality for DoD throughout the software development lifecycle.

3.17 Towards a Focus on Technical Debt Decision Making

Carolyn Seaman (University of Maryland – Baltimore, US)

My name is Carolyn Seaman and I am a professor and researcher in Baltimore, USA. I have been involved in funding, conducting, supervising, and publishing Technical Debt research since 2009. The area I am most interested in pursuing at this point is Technical Debt decision-making. Research and development on Technical Debt identification over the last 15 years has yielded a host of ways for software practitioners to gather data on the Technical Debt that exists in their products, but many fewer ways to make effective decisions about what Technical Debt to pay off and when. However, it's hard to evaluate decision-making approaches. Evaluation often relies on historical decisions, but these cannot be assumed to be optimal. Technical Debt researchers need collaborators from disciplines that have studied decision-making, such as finance, economic decision-making, and psychology.

3.18 Software engineering is evolving; is Technical Debt Management?

Tushar Sharma (Dalhousie University – Halifax, CA)

Software engineering (SE), with a boost from AI models, is evolving rapidly. Developers are not only generating code, tests, and documentation using AI models but also adopting automated AI agents that perform various SE-specific tasks such as code review. But what about Technical Debt Management? Do we observe the evolution at the same pace? In my lightning talk, I briefly discuss aspects that are keeping up with the rapid evolution and aspects, including tools, techniques, and methods involved in Technical Debt Management that require a significant boost.

3.19 Towards a Multi-dimensional Measuring and Observational Perspective of Technical Debt

Guilherme Horta Travassos (UFRJ / COPPE – Rio de Janeiro, BR)

Technical Debt emerges from decisions or situations throughout the software development cycle, jeopardizing evolution (most development workflows depend on it) and software maintenance due to its significant impact on internal software quality. Therefore, its identification is cardinal to managing the software project's well-being and ensuring the product's long life. However, identifying Technical Debt, including architecture Technical Debt, is not straightforward. Different context variables and factors (peopleware, processes, products, business, etc.) influence Technical Debt Management (see is Technical Debt Management platform at https://tdmframework.labculturadigital.com.br/). Technical Debt goes beyond source code. As far as it has been discussed in the sessions, a lack of grounded indicators

can make it challenging to indicate Technical Debt precisely. Its multidimensionality reduces the possibility of collecting just a simple measure to level Technical Debt. Considering the multiple perspectives Technical Debt can be perceived, and assuming Technical Debt items influence their interpretation, it is necessary to investigate further how Technical Debt can be perceived and measured through combining objective, subjective and qualitative data. Many context variables and features can jeopardize this investigation, such as the lack of contextualization, traceability, explainability, and transparency. It justifies a synergic collaboration between software stakeholders, tools vendors, and researchers aiming to share data and knowledge on Technical Debt. Such a collaboration will support us in a multi-dimensional measuring and observational perspective regarding Technical Debt.

3.20 Technical Debt Seeds

Roberto Verdecchia (University of Florence, IT)

A common misconception is that all Technical Debt is bad, and should be therefore always repaid. We need to revisit the concept of Technical Debt in order to more clearly spell out why all not Technical Debt should be fixed, how treating all Technical Debt equally can be a dangerous practice, and how we could proceed towards measuring and managing only the Technical Debt which really matters. As a first step forward, we can start from some Technical Debt corner cases, with the goal of shedding light into some nuances of the phenomenon that seem too often ignored by current practice and literature. Not all Technical Debt items should be addressed immediately, and some items should not be fixed at all. At the end. Technical debt is often associated to a risk. Evaluating such risk, rather than placing all Technical Debt in a bucket of items to be fixed, can lead to more accurate and effective Technical Debt prioritization and management strategies. A crucial question we should ask ourselves to advance current Technical Debt practices is therefore: How can we systematically evaluate which Technical Debt is most important?

3.21 IT Managers' Perspective on Technical Debt Management

Marion Wiese (Universität Hamburg, DE)

Joint work of Marion Wiese, Klara Borowa
Main reference Marion Wiese, Klara Borowa: "IT managers' perspective on Technical Debt Management", J. Syst. Softw., Vol. 202, p. 111700, 2023.
URL https://doi.org/10.1016/J.JSS.2023.111700

Technical Debt is a term for software solutions that are beneficial in the short term but impede future change. Previous research on Technical Debt indicates various management-related causes. We analyzed the perspective of IT managers on Technical Debt since they usually have a major influence on deadlines, the project's budget, and setting up a Technical Debt Management process. We obtained and analyzed data from 16 semi-structured interviews and a three-person focus group discussion. We found that all IT managers understood the Technical Debt concept. They considered Technical Debt Management to be an essential

topic, though almost none of them had set up a Technical Debt Management process so far. We identified three major IT managers' concerns regarding Technical Debt Management: (1) communicating about Technical Debt with business managers as well as developers, (2) establishing a Technical Debt Management process as IT managers seem to be missing guidance on how to introduce a sustainable Technical Debt Management process, and (3) dealing with vintage systems, i.e., old potentially debt-ridden but still valuable systems. In this work, we additionally developed a model specifying causes and consequences visible to business stakeholders, causal chains, and vicious cycles. The V4CTD model of Visibility, Cycles & Chains of Causes & Consequences of Technical Debt extends the Technical Debt conceptual model and facilitates communication on Technical Debt with business stakeholders.

Participants

- Apostolos Ampatzoglou
University of Macedonia – Thessaloniki, GR
- Paris Avgeriou
University of Groningen, NL
- Lodewijk Bergmans
Software Improvement Group – Amsterdam, NL
- Markus Borg
CodeScene – Malmö, SE
- Alexandros Chatzigeorgiou
University of Macedonia – Thessaloniki, GR
- Marcus Ciolkowski
QAware – München, DE
- Zadia Codabux
University of Saskatchewan – Saskatoon, CA
- Stefano Dalla Palma
Adyen – Amsterdam, NL
- Florian Deißenböck
CQSE – München, DE
- Philippe-Emmanuel Douziech
CAST – München, DE
- Neil Ernst
University of Victoria, CA
- Daniel Feitosa
University of Groningen, NL
- Michael Felderer
DLR – Köln, DE
- Collin Green
Google – Mountain View, US
- Ciera Jaspan
Google – Mountain View, US
- Ron Koontz
The Boeing Company – Mesa, US
- Heiko Koziolek
ABB – Mannheim, DE
- Christof Momm
SAP – Dresden, DE
- Brigid O'Hearn
Carnegie Mellon University – Arlington, US
- Ipek Ozkaya
Carnegie Mellon University – Pittsburgh, US
- Klaus Schmid
Universität Hildesheim, DE
- Carolyn Seaman
University of Maryland – Baltimore, US
- Tushar Sharma
Dalhousie University – Halifax, CA
- Guilherme Horta Travassos
UFRJ / COPPE – Rio de Janeiro, BR
- Roberto Verdecchia
University of Florence, IT
- Marion Wiese
Universität Hamburg, DE