Quantifying the Value of Technical Debt Removal: A Proposed Model

Sylvie Trudel[1,*,†], Donatien Koulla Moulla[2,†], Ria Bakhtiani[3,†], Thomas Fehlmann[4,†], Frank Vogelezang[5,†], Hassan Soubra[6,†] and Shashank Patil[3,†]

[1] *Université du Québec à Montréal (UQAM), Montreal, Canada*
[2] *University of South Africa, The Science Campus, Florida, 1710, South Africa*
[3] *Capgemini, 2 Lal Bahadur Shastri Marg, 400079 Mumbai, India*
[4] *Euro Project Office, Giblenstrasse 50, 8049 Zürich, Switzerland*
[5] *COSMIC, Canada/The Netherlands*
[6] *Ecole Centrale d'Electronique-ECE Lyon, 24 rue Salomon Reinach, 69007 Lyon, France*

Abstract
Technical debt hinders many organizations from improving their software to meet user demands. Opportunity: Therefore, it is important to have a model to quantify the value of technical debt removal. Methodology: The initial model was defined as the result of a brainstorming session with a team of experts. The model has then evolved from existing standards (literature), including functional size measurements (COSMIC). Results: In this position paper, we propose a model for quantifying the value of technical debt (TD) removal and define angles for operationalizing future research. This value should be quantified into the effort required to remove TD, considering the type of TD. The model includes information to support the prioritization of TD removal activities based on their expected value, including i) partners and vendors, ii) information and technology, iii) value stream and processes, and iv) organization and people.

Keywords
Technical debt removal, Estimation model, Maintainability, Defect density

1. Introduction

Technical debt has become a major challenge for many software organizations, hindering their ability to effectively meet evolving user demands and market needs. Ward Cunningham first introduced the Technical Debt (TD) metaphor in 1992 [1] in the following way: *"Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs*

IWSM-MENSURA'24: the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, September 30– October 4, Montreal, Canada
*Corresponding author.
†These authors contributed equally.
trudel.s@uqam.ca (S. Trudel); moulldk@unisa.ac.za (D. K. Moulla); ria.herrmann-bakhtiani@capgemini.com (R. Bakhtiani); thomas.fehlmann@e-p-o.com (T. Fehlmann); frank.vogelezang@cosmic-sizing.org (F. Vogelezang); hsoubra@ece.fr (H. Soubra); shashank.patil@capgemini.com (S. Patil)
https://capgemini.com/ (R. Bakhtiani); https://capgemini.com/ (S. Patil)
0000-0002-0877-7063 (S. Trudel); 0000-0001-6594-8378 (D. K. Moulla); 0000-0001-8257-825X (T. Fehlmann); 0000-0003-0056-6015 (H. Soubra); 0000-0003-1960-0420 (S. Patil)

when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise". While such practices may provide short-term benefits, like understanding customer behavior, they often lead to long-term consequences, such as increased complexity, decreased maintainability, and a higher likelihood of system failures or security vulnerabilities. [2, 3] The negative impact of technical debt on software development processes can be far-reaching. It can slow development velocity, as developers spend more time debugging issues and working around technical limitations rather than delivering new features. [4] This paper focuses on the technical debt that is causing a negative impact.

As technical debt accumulates, it becomes increasingly difficult and costly to address, creating a vicious cycle that can hinder an organization's ability to innovate and remain competitive. It is, therefore, important for software organizations to proactively manage and remove technical debt through dedicated refactoring efforts, architectural improvements, and the adoption of best practices in software development and maintenance [5, 6]. Quantifying the value of technical debt removal is a critical step in this process, as it allows organizations to prioritize their efforts, allocate resources effectively, and make informed decisions about when and how to address specific types of technical debt. By understanding the potential benefits of technical debt removal, such as increased development efficiency, improved code quality, and reduced maintenance costs, organizations can better justify and plan for these activities. [7]

This study proposes a model for quantifying the value of technical debt (TD) removal and defines angles for operationalizing future research. The model resulted from a workshop on technical debt removal at the 2023 IWSM-Mensura conference. This value should be quantified into the effort required to remove TD, considering the type of TD. The model includes information to support the prioritization of TD removal activities based on their expected value, including i) partners and vendors, ii) information and technology, iii) value stream and processes, and iv) organization and people.

Technical debt can evolve from four quadrants, coined by Martin Fowler, that combine either reckless or prudent behavior and is introduced either deliberately or inadvertently. [8] These quadrants help team members and stakeholders understand the different types of debt that can accumulate in the codebase. Types of technical debt need to be considered, as some types are easier to fix than others. This categorization of technical debt leads to categories like:

- Coding errors
- Business requirement evolution
- Design patterns
- Non-functional requirements (e.g. maintainability, security, stability, high availability)

To establish the value of technical debt removal, an estimation model is needed that contains at least the following elements:

- Cost of removal of technical debt: Cx
- Cost of non-removal of technical debt: Cy

With this estimation model, the benefits of technical debt can be quantified where Cy—Cx > 0. Technical debt removal is beneficial only if the benefits compensate for Cx. If there are multiple options for technical debt removal, those options can be prioritized.

What we want to achieve is to quantify technical debt and efforts for technical debt removal, using standard algorithms to express the value. To be able to classify existing research, we created the model in Figure 1 where the value of technical debt removal (Section 3) is influenced or supported by four angles that will be discussed in this paper:

- Organization and people (section 4)
- Partners and vendors (section 5)
- Information and technology (section 6)
- Value streams and processes (section 7)

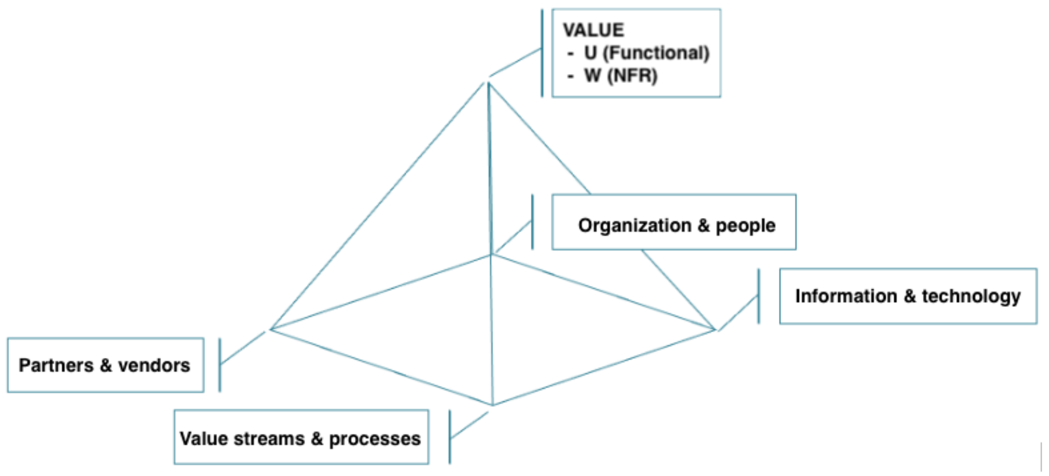

Figure 1: Value of Technical Debt Removal Model

In the following sections, we discuss for each angle of the model the following aspects:

- How can the (influence on) value be expressed?
- What methodologies can be used?
- What results do we expect?
- What research is needed?

2. Identification of Technical Debt in Agile mode

In Agile projects, we can assume the team has a formal ”Definition of Done" (DoD). In that case, any incomplete DoD element will lead to identifying a piece of the software’s technical debt. [9] Many teams will adopt a ”Definition of Ready" (DoR) for requirements, which includes a list of requirements characteristics to be considered ready to be developed. This means that whenever development is done without the requirements being considered ready, rework will most likely be required once the requirements are clarified or completed. Identification of technical debt within the requirements also comes from those elements of the DoR that have not been applied. However, technical debt may very well exist without defining DoD and DoR or from incomplete DoD or DoR.

3. Value

Expressing the value of technical debt removal involves assessing its impact on product quality, development speed, and maintenance efforts. Estimating the cost of delaying technical debt removal considers increased development time, risk of system failures, and missed opportunities. Reduction should be proactive, scheduled regularly, aligned with major releases, and prioritized based on impact. Continuous improvement fosters a culture of addressing debt iteratively or continuously [10].

The value of self-admitted technical debt removal refers to the deliberate effort within a software development team to address and eliminate known technical debt. Technical debt is the accumulated cost of additional rework caused by choosing an easy, expedient solution now instead of using a better approach that would take longer to develop at first. When a team acknowledges its incurred technical debt, it recognizes that certain aspects of its codebase or development processes need improvement. These could include outdated libraries, inefficient algorithms, poorly designed architecture, lack of documentation, or any other aspect hindering future development or maintenance.

Removing self-admitted technical debt involves allocating time and resources specifically to address these issues. This might include refactoring code, updating dependencies, improving test coverage, enhancing documentation, or revising development processes to prevent future debt accumulation. By proactively addressing technical debt, teams can improve their software products' quality, maintainability, and sustainability over time. This requires balancing the delivery of new features and allocating time for ongoing maintenance and improvement [11].

Aspects to be quantified:

- How can the value be expressed?
- What methodologies can be used?
- What results do we expect?
- What research is needed?

3.1. Functional Size Viewpoint on Value

In a well-monitored software development process [12, 13], the volume of functional size created, modified, or deleted for each sprint is a known quantity. If the development team employs a measurement method that enables the identification of base functional components (BFC) – for instance, data movements [14] – it is possible to mark those BFCs that are subject to technical debt. Identification occurs either when the functionality passes the DoD or the DoR under reserve of some known technical debt.

When using functional size measurements to quantify the value of technical debt that is present in developed software, it should be noted that measurement methods should be used in accordance with the International Vocabulary of Metrology [15] and should respect proper uncertainty measurement [16] to prevent inaccurate results. The introduction of additional uncertainties into the measurement process results in a lack of clear correlation between the size measurements of software components counted as technical debt and the overall size of the product.

Accordingly, the assessment of technical debt represents an integral aspect of the standard measurement process, occurring at each sprint. It is recommended that each sprint undergoes functional size measurement, including technical debt, given that some of the technical debt can be eliminated in a subsequent sprint. Furthermore, if the agile team identifies an accumulation of technical debt, inserting a technical debt removal sprint into the development process is always a viable option, contingent upon time availability.

The technical debt measurement is based on the functional size, which allows for direct comparison with the total planned or delivered functional size. It should be noted that the technical debt does not contribute to the overall size of the product. However, it may significantly impact the required effort for subsequent development, enhancement, or maintenance of the software product.

3.2. Non-Functional Size Viewpoint on Value

On the other hand, the non-functional viewpoint on the value of technical debt considers its impact on non-functional aspects of the software, such as reliability, performance, security, and maintainability. Technical debt can undermine these non-functional qualities, leading to increased downtime, decreased system responsiveness, heightened vulnerability to security threats, and greater difficulty in maintaining and evolving the software over time. From this perspective, addressing technical debt is essential for ensuring that the software meets quality standards, complies with regulatory requirements, and delivers a positive user experience.

Lastly, the most common method for estimating technical debt involves various approaches focusing on different aspects of technical debt quantification. These approaches include identifying smells, quantifying the Return on Investment (ROI) of refactoring, comparing the ideal state with the current state of software quality, and evaluating alternative development paths to reduce technical debt [17]. However, the lack of consistency among existing tools at the methodology and rule sets levels has made it challenging to effectively compare and evaluate these different estimation methods [18]. To address this issue, a novel conceptual model called the Technical Debt Quantification Model (TDQM) has been developed, which captures essential concepts related to technical debt quantification and allows for comparisons and evaluations between different approaches [19].

3.3. Automation

3.3.1. Automated measurement of Technical Debt

Automated Technical Debt (TD) measurement is crucial for effective software development management [20]. Existing studies highlight the complexity of TD measurement due to various aspects involved, such as code size, complexity, and modularity [21]. Self-Admitted Technical Debt (SATD) is a significant component of TD, often reported by developers themselves [22].

To address this, an approach integrating multiple sources like source code comments, commit messages, pull requests, and issue tracking systems has been proposed, achieving an F1-score of 0.611 in identifying different types of SATD [7]. However, the field of TD measurement is still evolving, with limited independent validation of models and few automated tools available. Further research is needed to enhance automated TD measurement processes and tools for efficient software development management.

3.3.2. Automatic technical debt estimation

Automatic technical debt estimation is a crucial aspect of software development, with existing literature highlighting various approaches to measuring technical debt principal. However, many current methods lack full automation, availability, and validation, leading to confusion among practitioners [23]. A study has shown that manual labelling and machine learning techniques can significantly improve the identification and estimation of technical debt within software projects [24]. For instance, machine learning models can be trained to estimate the severity of architectural smells, enabling the calculation of technical debt principal based on code analysis and prediction transparency. By leveraging these advancements, practitioners can better understand and address technical debt in their projects efficiently and accurately.

4. Organization and People

The three main sources of technical debt are [25]:

1. documentation,
2. wrong architecture, and
3. bad coding practices.

All three of them are highly dependent on the organization in which the software is developed and maintained and the people who develop and maintain the software.

4.1. Documentation

As mentioned in Section 2, in Agile projects, we can assume the team has a formal ”Definition of Done" (DoD). Any incomplete DoD element will lead to identifying a piece of the software's technical debt. When the organization is not facilitating the enforcement of the DoD by the scrum master, technical debt is more likely to exist and harder to fix.

Both specifications as solution documentation can be far away from code and obsolete before coding begins, or at least after finishing the work, describing an earlier idea of the solution approach. The ”Definition of Ready" (DoR) might have been applied at first, but as the code evolved, many teams did not update the specifications or the solution documentation, which then became obsolete. Identification of technical debt within the requirements also comes from those elements of the DoR that have not been applied. Tools are available to maintain documentation near the code, for instance, in the GIT realm. However, what if not properly used? Can obsolescence of the documentation be avoided? What if key people maintaining documentation or writing code are on leave temporarily?

4.2. Wrong architecture

As the software evolves over time, from maintenance requests, bug fixing and new business needs along with their related requirements (e.g. new functionalities), the presence of technical debt within the software architecture could be revealed through the observation that it takes more and more time and effort to make a change. When the software architecture was first defined, based on known requirements, it might have been adequate back then. However, after new needs emerged, it is possible that these new needs significantly impacted the architecture. In that case, not redefining the architecture will likely result in a lower development velocity.

4.3. Bad coding practices

However, even with the right documentation and the right architecture, technical debt may very well exist when bad coding practices are applied to the code. It is very important that processes or tools are in place to verify that the code that is produced adheres to quality standards and maintainability rules [26].

But what if time pressure opposes such a practice? Technical debt can be tolerated on purpose, but if it is left consciously, it should be managed.

5. Partners and Vendors

It is imperative that software procured from external sources, including partners, vendors, and open-source initiatives, undergo the same rigorous assessment of technical debt as outlined in section 4. This can be accomplished through contractual agreements with partners and vendors, or through rigorous incoming source inspection, or a combination of both. It is recommended that contracts and inspection practices specify the acceptable performance indicators for the quality of code, ideally supported by an independent tool that supplies quantified values for the different topics (such as code smells, vulnerabilities, code duplication, etc.). Furthermore, it is essential to clearly define and assess the documentation requirements (expected vs. delivered).

It is similarly crucial to achieve consensus on the procedure to be followed in the event that technical debt needs to be removed at a later stage, following acceptance and release. In the event that the organization responsible for the creation of the software is tasked with the removal of technical debt, it must be granted access to the relevant code and a clear understanding of the respective roles and responsibilities in the event of a failure must be established. If, on the contrary, the source code is retained by partners and vendors, it is of the utmost importance that all parties involved in the software development process reach a consensus and coordinate their measurements of technical debt. There should be a commonly agreed measurement process, see the respective section of this paper.

6. Information and Technology

The IT landscape is perpetually evolving, encompassing implementation methods and overall structure changes. Whether it's the ascendancy of automation, the emergence of low-code/no-code solutions, or the current rise of Gen-AI, each represents a significant shift in technology development and application. Amidst these transformations, one constant persists: the presence of technical debt.

Code produced through Gen-AI as shortcuts could accumulate as technical debt, requiring extensive rework to rectify issues or comply with evolving regulations and ethical standards. An organization needs to have a check and balance mechanism, ensuring any shortcut or Gen-AI is fit for use and preventing technical debt addressing ethical considerations or potential biases in the algorithms produced through Gen-AI.

During the Pursuit stage, the role and grade pyramid is experiencing a leftward shift due to Gen-AI, with junior resources being tasked with critical deliveries, relying more on Gen-AI code. This could potentially lead to the accumulation of technical debt in the future.

7. Value Stream and Process

Technical debt impacts the software development value stream from the initial development to deployment and maintenance. This leads to slower development speed as developers spend more time debugging issues and working around technical limitations. Consequently, technical debt creates a domino effect throughout the software development value stream. It slows down development, hinders quality, and makes deployment and maintenance more challenging. Removing technical debt as quickly as possible allows for improved development efficiency, code maintainability and streamlined deployment and maintenance.

To better understand the impacts of technical debt on the value stream, these aspects need to be quantified:

- How can the value be expressed? An organization may choose to express the value in terms of effort, duration or money;
- What methodologies can be used? As it is easier to improve an existing process (e.g. a documented process), continuous improvement activities are expected as a means to evolve the DevOps activities as part of the methodology as the basis of the value stream, but methodologies should also include measurement methods for base measures and measurement functions for derived measures;
- What results are expected? When the expected measurement results are understood, it facilitates the decisions about how these results will be communicated;
- What further research is needed? This position paper provides a basis to allow researchers to further investigate more precise ways to quantify technical debt removal.

8. Conclusion and Future Work

Technical debt remains a significant obstacle for many organizations striving to meet evolving user demands and maintain a competitive edge in the software industry. Our study has proposed a model for quantifying the value of technical debt removal, which provides a comprehensive framework for understanding and addressing the multifaceted nature of technical debt. By integrating elements from the Definition of Done (DoD) and Definition of Ready (DoR) within Agile methodologies, and considering both functional and non-functional aspects, our model offers a structured approach to identify, prioritize, and manage technical debt.

The model proposed in this paper emphasizes the importance of proactive technical debt management, suggesting that regular and systematic removal efforts aligned with major releases can substantially mitigate the negative impacts of technical debt. It highlights the necessity for organizations to balance new feature development with continuous improvement practices, thus fostering a sustainable development environment. The model also incorporates considerations from various perspectives, including organization and people, partners and vendors, information and technology, and value streams and processes, ensuring a holistic view of technical debt management.

This paper also presents the critical role of automation in measuring and estimating technical debt. Automated tools and machine learning techniques can significantly enhance the accuracy and efficiency of technical debt quantification, enabling teams to better understand the scope and severity of their technical debt. The integration of Self-Admitted Technical Debt (SATD) from multiple sources, such as source code comments and commit messages, provides a more comprehensive understanding of the debt landscape.

Despite the robustness of our proposed model, it is crucial to acknowledge that the software development landscape is constantly evolving. Emerging technologies, such as low-code/no-code platforms and Gen-AI, introduce new dimensions of technical debt that our current model may not fully address. Additionally, the dynamic nature of software requirements and market demands necessitates continual reassessment and adaptation of technical debt management strategies.

Therefore, while our model offers a solid foundation for quantifying and managing technical debt, it should not be viewed as exhaustive. Future research should focus on extending the model to incorporate these evolving technologies and their unique contributions to technical debt. Additionally, empirical validation and refinement through real-world case studies will be essential to ensure its applicability and effectiveness across diverse organizational contexts. By expanding our model to adapt to these emerging trends and continuous feedback, we can better equip organizations to manage their technical debt and enhance their software development processes.

References

[1] Cunningham, The wycash portfolio management system, in: Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum), OOPSLA '92, Association for Computing Machinery, New York, NY, USA, 1992, p. 29–30. URL: https://doi.org/10.1145/157709.157715. doi:10.1145/157709.157715.

[2] R. Nord, I. Ozkaya, 10 years of research in technical debt and an agenda for the future, Carnegie Mellon University, Software Engineering Institute's Insights (blog), 2022. URL: https://insights.sei.cmu.edu/blog/10-years-of-research-in-technical-debt-and-an-agenda-for-the-future/, accessed: 2024-Jul-10.

[3] E. Tom, A. Aurum, R. Vidgen, An exploration of technical debt, Journal of Systems and Software 86 (2013) 1498–1516. URL: https://www.sciencedirect.com/science/article/pii/S0164121213000022. doi:https://doi.org/10.1016/j.jss.2012.12.052.

[4] T. Adefioye, Managing Technical Debt, Apress, Berkeley, CA, 2023, pp. 149–179. URL: https://doi.org/10.1007/978-1-4842-9654-7_6. doi:10.1007/978-1-4842-9654-7_6.

[5] E. D. S. Maldonado, R. Abdalkareem, E. Shihab, A. Serebrenik, An empirical study on the removal of self-admitted technical debt, in: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017, pp. 238–248. doi:10.1109/ICSME.2017.8.

[6] F. Zampetti, A. Serebrenik, M. Di Penta, Was self-admitted technical debt removal a real removal? an in-depth perspective, in: 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR), 2018, pp. 526–536.

[7] Y. Li, M. Soliman, P. Avgeriou, Automatic identification of self-admitted technical debt from four different sources, Empirical Software Engineering 28 (2023) 65.

[8] R. Fowler, Technical debt quadrant, Martin Fowler (blog), 2009. URL: https://martinfowler.com/bliki/TechnicalDebtQuadrant.html/, accessed: 2024-Jul-12.

[9] S. Madan, Done: Understanding the definition of 'done', Scrum.org (blog), 2019. URL: https://www.scrum.org/resources/blog/done-understanding-definition-done, accessed: 2024-Jul-12.

[10] M. F. Harun, H. Lichter, Towards a technical debt management framework based on cost-benefit analysis, in: Proceedings of the 10 th International Conference on Software Engineering Advances, ????

[11] B. Pérez, C. Castellanos, D. Correal, N. Rios, S. Freire, R. Spínola, C. Seaman, C. Izurieta, Technical debt payment and prevention through the lenses of software architects, Information and Software Technology 140 (2021) 106692.

[12] T. Fehlmann, E. Kranich, A new approach for continuously monitoring project deadlines in software development, in: Proceedings of the 2017 IWSM Mensura conference, COSMIC, 2017.

[13] T. Fehlmann, E. Kranich, Functional size measurement in agile, in: Proceedings of the 2021 ICSEA conference, IEEE, 2021.

[14] A. Abran, COSMIC Measurement Manual for ISO 19761 – Version 5.0 – Part 1-3, Technical Report, 2020.

[15] ISO/TMBG, ISO/IEC Guide 99, International vocabulary of metrology – Basic and general concepts and associated terms, Technical Report, 2007.

[16] ISO/TMBG, ISO/IEC CD Guide 98-3, Evaluation of measurement data - Part 3: Guide to uncertainty in measurement, Technical Report, 2015.

[17] U. Vora, Measuring the technical debt, in: 2022 17th Annual System of Systems Engineering Conference (SOSE), IEEE, 2022, pp. 185–189.

[18] M. Mathioudaki, D. Tsoukalas, M. Siavvas, D. Kehagias, Comparing univariate and multivariate time series models for technical debt forecasting, in: International Conference on Computational Science and Its Applications, Springer, 2022, pp. 62–78.

[19] J. Perera, E. Tempero, Y.-C. Tu, K. Blincoe, Quantifying technical debt: A systematic mapping study and a conceptual model, arXiv preprint arXiv:2303.06535 (2023).

[20] I. Khomyakov, Z. Makhmutov, R. Mirgalimova, A. Sillitti, Automated measurement of technical debt: A systematic literature review, in: ICEIS 2019-Proceedings of the 21st International Conference on Enterprise Information Systems, 2019, pp. 95–106.

[21] A. Melo, R. Fagundes, V. Lenarduzzi, W. Santos, Identification and measurement of technical debt requirements in software development: a systematic literature review, arXiv preprint

arXiv:2105.14232 (2021).

[22] L. Capitán, B. Vogel-Heuser, Metrics for software quality in automated production systems as an indicator for technical debt, in: 2017 13th IEEE Conference on Automation Science and Engineering (CASE), IEEE, 2017, pp. 709–716.

[23] D. Sas, P. Avgeriou, An architectural technical debt index based on machine learning and architectural smells, IEEE Transactions on Software Engineering (2023).

[24] R. Sangwan, Automatically detecting technical debt discussions, Authorea Preprints (2023).

[25] E. Ramirez, La gestion de la dette technique (Technical Debt Management), Technical Report, Journée Francophone des Femmes en Informatique, April 29-30, 2024, Montreal (Canada) https://jffi.ca, 2024.

[26] J. R. Lahti, A.-P. Tuovinen, T. Mikkonen, Experiences on managing technical debt with code smells and antipatterns, in: 2021 IEEE/ACM International Conference on Technical Debt (TechDebt), IEEE, 2021, pp. 36–44.