Towards the Use of a Framework to Make Technical Debt Visible

María de León-Sigg, Sodel Vázquez-Reyes, Daniel Rodríguez-Ávila
Ingeniería de Software, Universidad Autónoma de Zacatecas
Zacatecas, México
mleonsigg@uaz.edu.mx, vazquezs@uaz.edu.mx, daniel.rodriguezav@gmail.com

***Abstract*— Technical debt concept has been in use since the 90s's decade. Several processes, techniques and tools, such as those related with software maintenance and risk control, are used to manage, prevent, measure and reduce technical debt. Technical debt management includes activities to identify, measure, prioritize, repay, and monitor it, but one of the main issues related with management resides in the complexity to make technical debt visible to organizations. In this paper is presented the application of the Normative Process Framework to make technical debt visible with a large system developed by students of software engineering. The Normative Process Framework is used in conjunction with a process to find technical debt and document it in a simple format. Results show how technical debt was made visible for that system in a simplified way, by using documentation generated during development, and considering not only code, but also other software assets. Once technical debt is made visible is easier to evaluate and prioritize it, to establish a convenient set of actions to control it.**

Keywords: technical debt, technical debt visibility, framework, technical debt management.

I. Introduction

In order to face the challenge to deliver products that satisfy customer needs, under tight time and resource constraints, software engineers sometimes have to reinterpret the approach they used to develop software [1]. This frequently means they have to favor short-term decisions related with design and implementation, under the risk that these decisions will imply higher costs in maintenance or even impossible changes in the future [2]. The focus on functionality over the use of sound practices in design, programming and testing, means that maintainability is not a main software quality driver [3]. The short-term benefits obtained are then achieved at the cost of long-term costs, which are not always well managed [4], [5]. Making an analogy to financial terms, this means that debt is generated, and that this debt should be paid with interests in the long-term. This analogy has served to define this situation with the term "technical debt" [2], [6], [4]. Technical debt can be introduced intentionally when developers are forced to create solutions with poor quality in order to meet stakeholder's deadlines, but also unintentionally, when developers lack of enough experience, there is a market need that forces software changes, teams fail to communicate adequately and software quality is reduced, or there is legacy software [7]. Technical debt is perceived when there is a slowdown in development or unusual alternative solutions are proposed or implemented [2].

As can be seen, technical debt can exist in almost every software product [8], however, when technical debt is not correctly managed, exists a high risk of the presence of negative consequences. Some of these include costly overruns, quality issues, inability to add new features without affecting existing ones, and the possibility of software becoming unusable sooner than planned [9], [4].

The concept of technical debt, conceived as those constructs in design or code, expedient in the short-term, but that imply costly or impossible future software evolution [10], has been present for a while [11], but until agile methodologies for software development become widely used, technical debt implications were really visible [12]. Since then, processes, techniques and tools, such as those related with software maintenance and risk control, are used to manage, prevent, measure and reduce technical debt during development [2]. The use of these approaches is known as technical debt management, and it is essential for the success of a project.

Technical debt management includes activities to identify, measure, prioritize, repay, and monitor it [13], [14], [7]. In order to manage technical debt, several approaches have been proposed. Some of them focus on technical issues [15], [16], [17], [18], [19], [20], while others propose frameworks to deal with a broader set of elements that influence technical debt introduction, such as stakeholders, business perspective, quality assurance methods, and management and engineering processes involved [21], [22], [23], [1]. Considering that technical debt is the result of schedule pressures, lack of a programming standard, developer errors, production infrastructure, and software architecture [24], the use of frameworks that consider organizational processes besides source code are more suitable to manage technical debt [21]. However, it should not be forgotten that technical debt management is complicated because it is not easy to visualize, quantify, and track it [7]. As can be noticed, a first step to manage technical debt should be its visualization, because if it is visible, it is easier to start other management activities [25].

In this document, are presented the results of the application of the Normative Process Framework [21] to make technical debt visible. The Normative Process Framework integrates processes to manage technical debt with the software quality management processes standards prescribed by the Project Management Body of Knowledge guide [26], [21]. The framework provides a mean to document technical debt [24], making it visible and, thus, facilitating its subsequent quantification [21].

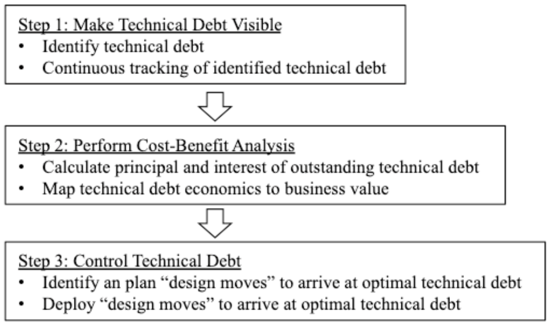
978-1-7281-8450-0/20/$31.00 ©2020 IEEE
DOI 10.1109/CONISOFT50191.2020.00022

This document is organized as follows: in the methodology section is explained how the Normative Process Framework was used and adapted. In the third section are presented the results obtained when applied to a previously developed product, and, in the fourth section, they are discussed. Finally, in the fifth section, the conclusions derived from this work are offered.

II. METHODOLOGY

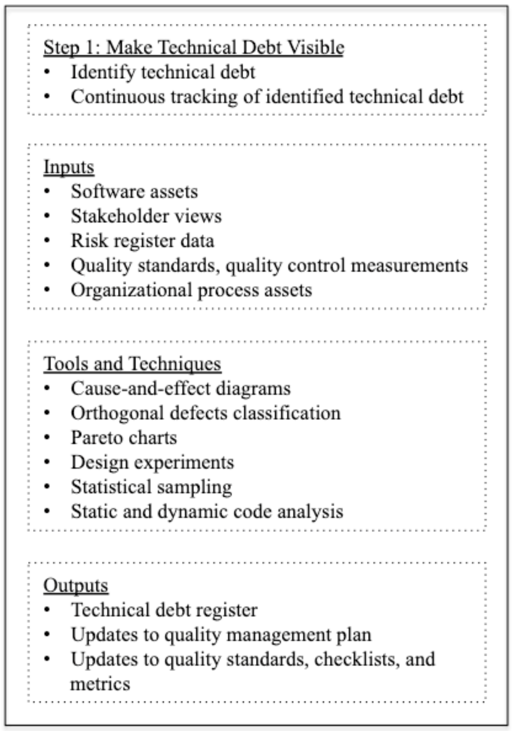
To make technical debt visible, was adapted the Step 1 of the Normative Process Framework, proposed by Ramasubbu and Kemerer [21]. The framework, shown in Fig 1., establishes three steps to managing technical debt: 1) Make technical debt visible; 2) Perform cost-benefit analysis; and 3) Control technical debt. Due to the scope of the research presented in this document, only Step 1, Make technical debt visible, was considered, to find technical debt during the maintenance of a software product. This product will be described later in this section.

In Fig. 2, are shown the inputs, tools and techniques and outputs of Step 1 of the Normative Process Framework [21]. From Fig. 2, can be noticed that the framework considers as inputs software assets, risk register data, and quality measurements, among others. Some tools and techniques recommended by framework are defect classification and cause-and-effect diagrams. And, finally, the outputs of Step 1 include the updates to quality management plan, quality standards, checklists and metrics.



Step 1: Make Technical Debt Visible
• Identify technical debt
• Continuous tracking of identified technical debt
Step 2: Perform Cost-Benefit Analysis
• Calculate principal and interest of outstanding technical debt
• Map technical debt economics to business value
Step 3: Control Technical Debt
• Identify an plan "design moves" to arrive at optimal technical debt
• Deploy "design moves" to arrive at optimal technical debt

Figure 1. Integrated process framework to manage technical debt (adapted from Ramasubbu and Kemerer [21])



Step 1: Make Technical Debt Visible
• Identify technical debt
• Continuous tracking of identified technical debt
Inputs
• Software assets
• Stakeholder views
• Risk register data
• Quality standards, quality control measurements
• Organizational process assets
Tools and Techniques
• Cause-and-effect diagrams
• Orthogonal defects classification
• Pareto charts
• Design experiments
• Statistical sampling
• Static and dynamic code analysis
Outputs
• Technical debt register
• Updates to quality management plan
• Updates to quality standards, checklists, and metrics

Figure 2. Step 1 inputs, tools and techniques, and outputs (adapted from Ramasubbu and Kemerer [21])

For the purposes of the work presented here, the inputs considered to make technical debt visible were the code, the design, the list of risks, the log of defects, the requirements' specification, the operation guide, and the user's manual. The technical debt was identified by one of the system's developers, using the technical debt evaluation proposed by Yli-Huumo et al. [25]. The evaluation consisted in the analysis of the answers to the questions: a) What are the benefits of fixing this issue?; b) Are there any risks in fixing this issue?; c) Why was this issue done previously like that?; and, d) How to fix this issue and what resources the fix would require?. As the output of the Step 1, it was obtained a technical debt register documented in a form that included an identifier of the technical debt found, as well as its description. During the product's maintenance, testing was used to verify functionalities, and as defects were found, they were corrected, to update the checklist and the metrics used during development, as is also depicted in Fig. 3.

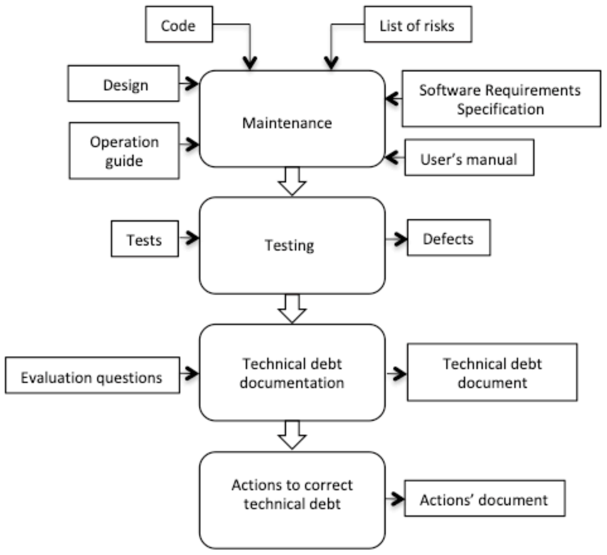


Figure 3. Activities to apply the Step 1 of the Normative Process Framework

Framework's Step 1 was applied during the maintenance of a software product developed by undergraduate students of software engineering enrolled in the course of Team Software Process. Students had prior training in software analysis and design, personal software process and development frameworks. The product was a system to manage social service activities of medicine interns. Social service coordinator was the system's acquirer. System's functionality included a social service students' register, a module to share and query information about social service legislation, a forum to share questions and answers about medicine topics, a message service between social service interns and social service coordinator implemented with a bot, a clinic catalog, an incident's report module, and a panic button to help interns under threat. Requirements were expressed using use cases, and a total of 108 use cases were implemented.

Because of its size, development was spread in four terms. In each term, a set of requirements was developed, accordingly with acquirer priorities, as shown in Table I. In this way, students enrolled in the first term developed the first set of requirements, and the developed system was then inherited to next term's course enrolled students, who refined it and added new required functionality. This procedure was repeated until the fourth development's term. Later, in a fifth term, the product was given maintenance and the framework's Step 1 applied.

TABLE I. DISTRIBUTION OF SYSTEM DEVELOPMENT

Term	**Functionality developed**	**Use cases implemented per term**	**Number of teams**
1	Coordinator, interns and clinics register module, information about social service legislation management module	28	4
2	Update to clinic register functions, panic button, incident reports management module	25	4
3	Experts register module, forum to answer questions, Update to panic button functions	31	4
4	Messages management module, clinic catalog module	24	6

In the four terms of development, the standard ISO/IEC 29110 Entry Profile [27] was used to guide the system realization as well as the project management. The execution of the standard's tasks generated the inputs used for Step 1 of the Normative Process Framework, as shown in Fig 3.

The software product maintenance objectives were its migration to a new server and the addition of functionality to access a list of frequently asked questions, and to access an external learning system. The person in charge of the maintenance was a student who helped develop the system in the last term of its development. Because of this, the student had extensive knowledge about the product development history. This knowledge allowed him to identify which documented risks during the product's development had become issues related with technical debt, and to answer the questions for the evaluation of the technical debt. During the product's maintenance, testing was used to verify functionalities, and as defects were found, they were corrected, as is also shown in Fig. 3. As technical debt was found, it was documented. Finally, actions done to correct some of the technical debt found were also registered.

III. RESULTS

To make technical debt visible, the risks found and monitored during development were considered. The risk's data document was generated during project planning, as it is stated in the task PM1.10 of ISO/IEC 29110 Entry Profile. An excerpt of those risks is presented in Table II. The risks indicated in the Table II were documented during the four terms of product's development, as well as their impact and the likelihood of them to occur. In Table II, the risk impact is weighted in a scale from 1 to 10, where 1 has little impact and 10 has large impact; and the likelihood of occurrence is also weighted in a scale from 1 to 10, where 1 is unlikely to happen and is 10 very likely to happen. Also, there were considered the system's design and software code, available to all the involved students during the four terms of development.

The system was subjected to different tests. Each test was documented with the next information:

TABLE II RISKS DETECTED

Risk ID	Risk	Impact	Likelihood
R-001	Missing documentation about existent system functionality	7	8
R-002	Poor feedback from system users	8	8
R-003	Functionality not considered for more than one user	7	5
R-004	Code no longer needed because of requirements change	5	4
R-005	Requirements change	7	6
R- 006	New functionality not considered during system development	8	6

- Test name
- Test objective
- Test description
- Test date
- Test conditions
- Expected results
- Actual results
- Test results

Table III shows some descriptions of the tests done to the system.

Each one of the tests had an objective. Some of the documented test's objectives were:

- Create users successfully
- Create nonconformity reports in a correct way
- Create forum questions correctly
- Test panic button functionality
- Modify user's data
- Upload normative documentation
- Assign interns correctly

To be assured that the information was gathered appropriately, there were defined some standards. The student doing the product's maintenance elaborated the definition of the standards. The identification as well as the description of some of them is shown in Table IV.

As mentioned in the methodology section, students had training in personal software process, so they recorded the defects found during the system's development, and the student who did the maintenance updated the system's defect log. The reason to do this was that defects are related with the evidence of problems not found during development, including the testing stage, or with deviations that generated technical debt. Besides, it is a reflection of the impact generated with defects not detected in early developments stages.

Also, it was documented the time needed to solve those defects, because time is directly related with costs, and time measurement of defects solving.

Both metrics can be justified because finding defects implies effort related with solving them. Also, they are related with product's requirements because if those defects affect the desired functionality, it could mean that the product was not reviewed adequately before deployment.

TABLE III TEST DESCRIPTION

Test ID	Test description
TC-001	As coordinator I want to test system login
TC-002	As coordinator, I want to upload normative information
TC-003	As coordinator, I want to upload normative information without a file
TC-004	As coordinator, I want to upload normative information without introduce required data
TC-005	As coordinator, I want to see normative information uploaded
TC-006	As coordinator, I want to register a new clinic
TC-007	As coordinator, I want to search for saved clinics
TC-008	As coordinator, I want to register a new clinic without obligatory data
TC-009	As coordinator, I want to obtain a pdf file report of clinics
TC-010	As coordinator, I want to obtain a xls file report of clinics
TC-011	As coordinator, I want to see clinics ordered by complement
TC-012	As coordinator, I want to see clinics ordered by name
TC-013	As coordinator, I want to see clinics ordered by location
TC-014	As coordinator, I want to see clinics ordered by status
TC-015	As coordinator, I want to search clinics for a criteria written in the search line
TC-016	As coordinator, I want to delete clinic information
TC-017	As coordinator, I want to activate/deactivate a clinic
TC-018	As coordinator, I want to modify clinic information
TC-019	As coordinator, I want to see expert registered
TC-020	As coordinator, I want to register a new expert
TC-021	As coordinator, I want to register a new tutor with incomplete data

TABLE IV STANDARD DEFINITION

Standard ID	Description
E-001	Every test was documented with information a name, and objective, a description, a set of conditions, expected results and actual results.
E-002	Every defect must have an identifier, a description and the time taken to solve them.
E-003	Every corrected or created method must have a description of its functionality.

This way, defects were identified and documented using a log with a defect ID, a defect description, and time used to solve it. An excerpt of defects found during maintenance is shown in Table V.

The student in charge of maintenance evaluated each defect, accordingly with the questions presented in methodology. The analysis of defect data, as well as the answers given to each of the evaluation questions, allowed the finding of technical debt. An excerpt of the technical debt found is listed in Table VI.

Time dedicated to achieving maintenance objectives was also recorded. This register would be useful to help clarify costs generated when there is not an adequate control of decisions made by development teams, and to update the quality management plan, as Step 1 of Normative Process Frameworks indicates in Fig. 2. An excerpt of this register is shown in Table VII.

TABLE V DEFECTS FOUND

ID Defect	Description	Time (min)
D-001	Defect when uploading a normative file	60
D-002	Defect to enter service provider page	72
D-003	Defect when finding clinics by status	33
D-004	Defect when selecting the clinic catalog as coordinator	20
D-005	Letters can be written when tester is trying to text a tutor's telephone	43
D-006	There is not a validation when trying to modify tutor's data	52
D-007	xls file is not generated when required with empty data	62
D-008	Defect when displaying messages	120

TABLE VI TECHNICAL DEBT FOUND

Technical debt ID	Description
DT-001	Lack of updated documentation on system functionality
DT-002	Lack of documentation on source code
DT-003	Low level of legibility in source code
DT-004	Missed scalability in user classification
DT-005	Lack of a deployment guide
DT-006	Lack of users feedback on system
DT-007	Poor implementation of messages section
DT-008	Lack of code standards

IV. DISCUSSION

As shown in the previous section, the application of the Step 1 of the Normative Process Framework to make technical debt visible as well as the use of a simple process is convenient to document technical debt. The experience of developers is useful to make technical debt visible, but the use of a process that considers as input previous documentation generated during development is a convenient way to establish simple practices that organizations could use to begin to manage technical debt.

As also was shown, technical debt documentation can be done with simple formats, that could be put into consideration of development teams when they are more or less used to be guided by international standards to develop and manage development.

The results obtained in this work come from the application of the Normative Process Framework in a single product developed by students to relate knowledge with real-life practice, after four terms of development. This is a limitation of the results obtained because the technical debt was introduced since the first term, but it was made visible after almost two years of development, so there are decisions made that could remain imperceptible during the process of making technical debt visible.

Other source of limitation can be found in the consideration that the Normative Process Framework and the process to apply it were only used with one product, so it is left to future research their use with other products.

V. CONCLUSIONS

Technical debt cannot be completely avoided when software development is focused on satisfy customer needs as soon as possible, with solutions suitable for the short-term. Because of this, it is critical to keep in mind that an efficient management of technical debt is needed.

TABLE VII TIME REGISTER IN MAINTENANCE

Action taken	Time (min)
Configuration of new server	122
Preparation of system environment	130
Rules to server connection	32
Defect solving	462
Add functionality of frequent questions	62
Add functionality to link an external learning management system	102

However, technical debt management is not easy because there are different factors that impact decision's making while software is being developed.

Due to this, a first step to manage technical debt is to make it visible, independently if it was introduced intentionally or unintentionally. The use of frameworks that consider other elements than the architecture and the code of a software product, results in an appropriate way to manage technical debt.

Then, the first step to manage technical debt is to make it visible and evident to the whole organization to be able to find actions to reduce it or eliminate it in a product, or, in the worst case, to learn to live with it.

Despite making technical debt visible could be considered a complicated task, in this paper was shown that it could be simplified with the use of documentation generated during the system's development as well as the use of a process to document technical debt during maintenance. Once technical debt is made visible, it is easier to evaluate and prioritize it, and, consequently, to establish a convenient set of actions to control it. The documentation of the respective process to do so, the processes needed to monitor actions' convenience, as well as the definition of metrics to compare effectiveness of processes in other academic development but also in industry developments, are future works derived from the one presented here.

REFERENCES

[1] J. Holvitie et al., "Technical debt and agile software development practices and processes: An industry practitioner survey," Inf. Softw. Technol., vol. 96, no. September 2016, pp. 141–160, 2018.

[2] P. Kruchten, R. Nord, and I. Ozkaya, Managing Technical Debt: Reducing Friction in Software Development, 1st editio. Pearson Education, 2019.

[3] P. Kruchten, R. Nord, and I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice," IEEE Softw., vol. 29, no. 6, pp. 18–21, 2012.

[4] T. Besker, A. Martini, and J. Bosch, "The Pricey Bill of Technical Debt - When and by whom will it be paid ?," in 2017 IEEE Internationa Conference on Software Maintenance and Evolution (ICSME), 2017, pp. 13–23.

[5] C. Fernández-Sánchez, J. Garbajosa, and A. Yagüe, "A Framework to Aid in Decision Making for Technical Debt Management," in IEEE 7th International Workshop on Managing Technical Debt (MTD), 2015, pp. 69–76.

[6] Y. Guo, R. . Spinola, and C. Seaman, "Exploring the Costs of Technical Debt Management: A Case Study," Empir. Softw. Eng., vol. 21, no. 1, pp. 159–182, 2016.

[7] J. Yli-huumo, A. Maglyas, and K. Smolander, "How Do Software Development Teams Manage Technical Debt ? – An Empirical Study," J. Syst. Softw., vol. 120, pp. 195–218, 2016.

[8] M. F. Harun and H. Lichter, "Towards a Technical Debt Management Framework based on Cost-Benefit Analysis," in The Tenth International Conference on Software Engineering Advances, 2015, no. November, pp. 70–73.

[9] C. Seaman et al., "Using technical debt data in decision making: Potential decision approaches," in 2012 3rd International Workshop on Managing Technical Debt, MTD 2012 - Proceedings, 2012, pp. 45–48.

[10] P. Avgeriou, P. Kruchten, R. L. Nord, I. Ozkaya, and C. Seaman, "Reducing friction in software development," IEEE Softw., vol. 33, no. January, pp. 1–14, 2016.

[11] W. Cunningham, "The WyCash Portfolio Management System," in Addendum to the Proceedings on Object-Oriented Programming System Applications, 1992, pp. 29–30.

[12] R. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas, "In Search of a Metric for Managing Architectural Technical Debt," in 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, 2012, pp. 91–100.

[13] P. Avgeriou, P. Kruchten, I. Ozkaya, C. Seaman, and C. Seaman, "Managing Technical Debt in Software Engineering (dagstuhl seminar 16162)," in Dagstuhl Reports, 2016, vol. 6, no. 4, pp. 110–138.

[14] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "The Financial Aspect of Managing Technical Debt : A Systematic Literature Review," Inf. Softw. Technol., vol. 64, pp. 52–73, 2015.

[15] N. Ramasubbu, C. F. Kemerer, and C. J. Woodard, "Managing technical debt: Insights from recent empirical evidence," IEEE Softw., vol. 32, no. 2, pp. 22–25, 2015.

[16] A. Martini, E. Sikander, and N. Madlani, "A Semi-Automated Framework for the Identification and Estimation of Architectural Technical Debt: A Comparative Case-Study on the Modularization of a Software Component," Inf. Softw. Technol., vol. 93, no. October, pp. 264–279, 2018.

[17] A. Martini and J. Bosch, "An empirically developed method to aid decisions on architectural technical debt refactoring," no. February 2018, pp. 31–40, 2016.

[18] C. Fernández-Sánchez, J. Garbajosa, C. Vidal, and A.

Yagüe, "An Analysis of Techniques and Methods for Technical Debt Management: A Reflection from the Architecture Perspective," Proc. - 2nd Int. Work. Softw. Archit. Metrics, SAM 2015, no. May, pp. 22–28, 2015.

[19] P. Kouros, T. Chaikalis, E. M. Arvanitou, and A. Chatzigeorgiou, "JCaliper: Search-Based Technical Debt Management," in 34th ACM/SIGAPP Symposium on Applied Computing, 2019, pp. 1721–1730.

[20] T. Chen, R. Bahsoon, S. Wang, and X. Yao, "To adapt or not to adapt? Technical debt and learning driven self-adaptation for managing runtime performance," in Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, 2018, vol. 2018-March, pp. 48–55.

[21] N. Ramasubbu and C. Kemerer, "Integrating Technical Debt Management and Software Quality Management Processes," in Proceedings of the 40th International Conference on Software Engineering, 2018, pp. 883–883.

[22] C. Fernández-Sánchez, J. Garbajosa, A. Yagüe, and J. Perez, "Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study," J. Syst. Softw., vol. 124, no. November 2017, pp. 22–38, 2017.

[23] A. Ampatzoglou, A. Michailidis, C. Sarikyriakidis, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "A framework for managing interest in technical debt," in Proceedings of Technical Debt 2018 International Conference, 2018, no. March, pp. 115–124.

[24] P. Kruchten, R. L. Nord, and I. Ozkaya, Managing Technical Debt: Reducing Friction in Software Development, 1st ed. Pearson Education, 2019.

[25] J. Yli-Huumo, A. Maglyas, K. Smolander, J. Haller, and H. Törnroos, "Developing Processes to Increase Technical Debt Visibility and Manageability. An Action Research Study in Industry," in International Conference on Product-Focused Software Process Improvement, 2016, pp. 368–378.

[26] R. R. de Almeida, C. Treude, and U. Kulesza, "Tracy : A Business-driven Technical Debt Prioritization Framework," in 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2019, pp. 181–185.

[27] ISO/IEC, "TECHNICAL REPORT ISO / IEC TR Software engineering — Lifecycle profiles for Very Small Entities (VSEs) — Management and engineering guide: Generic profile group: Entry profile," Geneva, Switzwerland, 2015.