Technical Debt in Continuous Software Engineering: An Overview of the State of the Art and Future Trends

Lucas de Oliveira Carvalho[1], **João Paulo Biazotto**[1,2], **Daniel Feitosa**[2], **Elisa Yumi Nakagawa**[1]

[1]University of São Paulo (USP), São Carlos, Brazil

[2]University of Groningen, Groningen, The Netherlands

Abstract. *Large software companies strive to make their engineering processes fast, and agile development has been a key enabler for flexible delivery of solutions following the market needs. In this context, continuous software engineering (CSE) has emerged as a way to iteratively develop software using practices that encompass business strategy, development, and operations that are aligned with the agile methodology. However, these practices can also lead to the accumulation of technical debt (TD), which has shown to be harmful to the software in the long-term. Due to its impact, TD should be managed in the context of CSE. However, to the best of our knowledge, there is a lack of an overview of how TD has been addressed in this context. In this study, we present the state of the art of TD in CSE; for this, we scrutinized the literature and found 41 studies. Our main findings show that this field of study is relatively new, with active participation of the industry, and that most CSE activities are not addressing TD yet; therefore, presenting a number of opportunities for future research.*

1. Introduction

Large software companies have attempted to make their development processes responsive and with minimal time between the identification of the market needs and the delivery to end users, and these attempts have found an ally with agile development [Dingsøyr et al. 2012]. In the last years, the software industry has widely adopted agile software development as a suitable way of engineering software systems. Agility has proved to be beneficial to the industry in various aspects, like software environments, technologies, requirements, user needs, and market changes [Papatheocharous and Andreou 2013, Vijayasarathy and Turk 2012].

In this context, Continuous Software Engineering (CSE) has emerged as a way to iteratively develop software using practices, such as continuous planning, continuous integration (CI), continuous deployment (CD), and continuous monitoring [Fitzgerald and Stol 2017]. As an instance, DevOps is a broadly used and well-recognized concept that has attracted much interest from both academia and industry because of its focus on leveraging the continuous delivery of software products to end users, matching then the current industry needs [Chen 2015]. CSE handles software engineering as a continuous flow and comprises three phases [Fitzgerald and Stol 2017]:

- Business Strategy: The operations feedback (e.g., end user needs, market changes, production metrics) is used to replan the software evolution aiming at aligning the development of new features with the business needs;

- Development: It covers the main activities of the software development process from analysis to maintenance. These activities also handle a stream of new features to be developed by applying techniques associated with continuous architecting, integration, deployment, verification, and others; and
- Operations: This phase focuses on the continuity of the software in terms of usage and runtime management. The companies need to ensure that the software systems deliver the values that fulfill the users' needs and also assure the reliability of these systems through runtime monitoring and dynamic adaptation over time.

Although agile software development is usually seen as beneficial, the short deadlines that are imposed by this process can lead to bad decisions. Such sub-optimal decisions tend to meet short-term goals (e.g., speeding up the development), but may lead negative impacts in the long-term (such as quality issues or development gaps), which is called technical debt (TD) [Kruchten et al. 2012]. Alves et al. [2016] highlight that TD can refer to various artifacts (e.g., code, test, documentation) or processes (e.g., build, requirements) in the software and its development. And, on this note, Seaman et al. [2012] illustrate several consequences if TD is not solved, such as large cost overruns, quality issues, inability to add new features without disrupting new ones, and the premature loss of the system. In summary, although TD is being increasingly discussed in the software community and being investigated as a phenomenon that can occur in any phase of the software life cycle, several aspects of it need to be still investigated, and some issues are still open for research.

While the concept of TD has existed for some time, the agile development adoption has given to it the known visibility [Codabux and Williams 2013]. Due to the impact of TD and the necessity of TD management, it should be evaluated in the context of CSE, specially with the companies and software projects adopting CSE activities in a daily basis [Klotins and Gorschek 2022]. However, to the best of our knowledge, there is a lack of an overview of how TD has been addressed in the CSE context.

In this scenario, the main contribution of this study is to present a rundown of the state of the art of how TD has been addressed in CSE. For this, we conducted a systematic mapping study (SMS) and selected 41 studies. The results of our SMS show that researching TD in CSE is relatively new, with studies distributed in the last 10 years; besides, some activities of CSE, specially continuous architecting, have addressed TD, while other fundamental activities (e.g., CI) have not yet and no study has addressed TD in the whole CSE cycle. Another important finding is that the industry has been considerably involved with the field, showing its interest in investigating the problem. For the future, further research is essential to progress and consolidate the TD management within the CSE activities and future perspectives are also presented.

This paper is organized as follows: Section 2 presents the research method and how it was planned and conducted; Section 3 presents the established state of the art; Section 4 discusses the findings, perspectives of future research, and threats to validity; and Section 5 concludes this work.

2. Research method

To conduct this SMS, we followed the guidelines proposed by [Petersen et al. 2008] and [Kitchenham et al. 2015]. Three main phases were followed to conduct this study: (i)

Planning: creation of the protocol, which addresses objective, research questions, search criteria and selection criteria of the studies; (ii) Conduction: selection of the studies based on the protocol; and (iii) Results: synthesis and description of the results. These phases (illustrated in Figure 1) are detailed in Sections 2.1, 2.2, and 2.3.

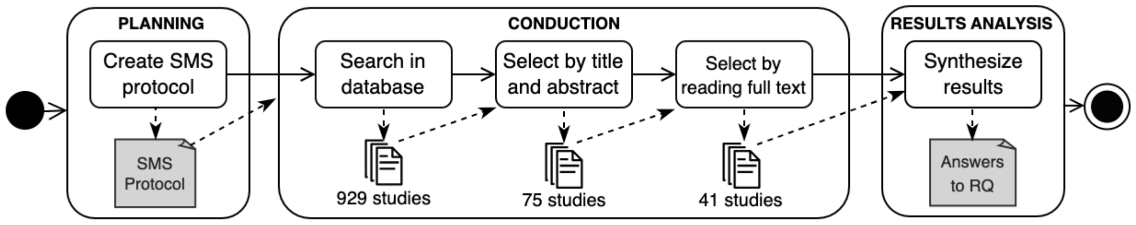

Figure 1. Phases for the conduction of this SMS.

2.1. SMS Planning

With the objective of uncovering the main characteristics of TD in CSE, this SMS addresses the following research question (RQ): *Which of the CSE activities address TD?*

The search strategy started by creating a search string that could collect the maximum amount of relevant studies.The main related terms were discussed by experts and were used in a pilot search to build a search string composed of the main terms related to TD and CSE. The final search string was:

("debt" AND ("continuous" OR "devops" OR "bizdev" OR "agile" OR "lean"))

Considering that Scopus[1] indexes more than 7,000 publishers[2], including the most relevant journals and conferences for the field of Software Engineering besides being widely adopted in SMS and SLR (systematic literature review) in this field, we decided to adopt only it to reveal the overview of the state of the art intended in this study. For this preliminary study that offers such an overview, we decided not to perform a snowballing-based review [Wohlin 2014].

In order to systematically filter the studies and select only the relevant ones to answer the RQ, one inclusion criterion (IC) and five exclusion criteria (EC) were established, as follows:

- IC1: Study addresses TD in CSE activities/environments (e.g., agile development, continuous integration, continuous delivery, etc);
- EC1: Study is not written in English;
- EC2: Study is a shorter version of another study already included;
- EC3: Study's full text is not available;
- EC4: Study is a conference proceedings, tutorial, interview, editorial, workshop, keynotes or summary of keynotes, preview or a preliminary study;
- EC5: Study is not a primary study;

[1]`https://www.scopus.com/`
[2]`https://www.elsevier.com/solutions/scopus/how-scopus-works`

2.2. SMS Conduction

Figure 1 summarizes the SMS conduction steps, also showing the number of selected studies at the end of each step. The SMS was conducted between August 2023 and November 2023 and split into three main steps:

- Search in database: The final search string created in the Planning phase was configured for the Scopus database to retrieve studies. The string was used to search studies through titles, keywords, and abstracts. At the end of this step, the total amount of primary studies returned was 929;
- Selection by title and abstract: The selection criteria settled at the Planning phase were applied by reading the title and abstract of each study. At the end of this step, 75 studies were selected;
- Selection by full-text reading: The selection criteria were applied once again after reading the full text of each study, and 41 studies were selected and are listed in Table 1.

2.3. SMS Synthesis

To synthesize the data collected from the studies, we extracted the information related to the RQ. Along with the authors, publication date, and publication venue, we also extracted: study type, relationship between academia and industry, application domain, types and/or sources of TD. The results uncovered in this study are presented in Section 3.

3. Results

Table 1 summarizes the selected studies after executing the steps discussed in Section 2. This table shows an identification and the reference for each study. We show them here instead of among the bibliographic references to facilitate the reading of the paper. Furthermore, supplementary material[3] presents the data synthesized in this section and discussed further in this work.

Table 1. Selected studies for the SMS.

ID	Reference
S1	Dos Santos, P. S., Varella, A., Dantas, C., and Borges, D. (2013). **Visualizing and managing technical debt in agile development: An experience report** page 14.
S2	Codabux, Z. and Williams, B. (2013a). **Managing technical debt: An industrial case study**. In 4th International Workshop on Managing Technical Debt (MTD), page 8 – 15
S3	Martini, A., Bosch, J., and Chaudron, M. (2014). **Architecture technical debt: Understanding causes and a qualitative model**. In 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, pages 85–92
S4	Holvitie, J., Lepp anen, V., and Hyrynsalmi, S. (2014). **Technical debt and the effect of agile software development practices on it - an industry practitioner survey**. In 2014 Sixth International Workshop on Managing Technical Debt, pages 35–42.
S5	Sandberg, A. B., Staron, M., and Antinyan, V. (2015). **Towards proactive management of technical debt by software metrics**. In Symposium on Programming Languages and Software Tools

Continued on next page

[3]https://doi.org/10.6084/m9.figshare.25481542

Table 1 – *Continued from previous page*

Id	Reference
S6	Martini, A. and Bosch, J. (2015b). **Towards prioritizing architecture technical debt: Information needs of architects and product owners**. In 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, pages 422–429.
S7	Soares, H., Rios, N., Mendes, T., Mendonc a, M., and Sp ınola, R. (2015). **Investigating the link between user stories and documentation debt on software projects**. Proceedings - 12th International Conference on Information Technology: New Generations, ITNG 2015, pages 385–390.
S8	Martini, A. and Bosch, J. (2015a). **The danger of architectural technical debt: Contagious debt and vicious circles**. In 2015 12th Working IEEE/IFIP Conference on Software Architecture, pages 1–10
S9	Martini, A., Bosch, J., and Chaudron, M. (2015). **Investigating architectural technical debt accumulation and refactoring over time: A multiple-case study**. Information and Software Technology, 67:237–253.
S10	Griffith, I., Taffahi, H., Izurieta, C., and Claudio, D. (2014). **A simulation study of practical methods for technical debt management in agile software development**. In Proceedings of the Winter Simulation Conference 2014, pages 1014–1025
S11	Mendes, T., Soares, H., Farias, M., Kalinowski, M., Mendonça, M., and Spínola, R. (2016). **Impacts of agile requirements documentation debt on software projects: A retrospective study/**
S12	Vathsavayi, S. H. and Syst a, K. (2016). **Technical debt management with genetic algorithms**. In 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pages 50–53.
S13	Vassallo, C., Zampetti, F., Romano, D., Beller, M., Panichella, A., Di Penta, M., and Zaidman, A. (2016). **Continuous delivery practices in a large financial organization**. In 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 519–528.
S14	Gupta, R. K., Manikreddy, P., Naik, S., and Arya, K. (2016). **Pragmatic approach for managing technical debt in legacy software project**. In Proceedings of the 9th India Software Engineering Conference, ISEC '16, page 170–176, New York, NY, USA. Association for Computing Machinery
S15	Martini, A. and Bosch, J. (2017a). **The magnificent seven: towards a systematic estimation of technical debt interest**. In Proceedings of the XP2017 Scientific Workshops, XP '17, New York, NY, USA. Association for Computing Machinery.
S16	Ciolkowski, M., Guzmán, L., Trendowicz, A., and Salfner, F. (2017). **Lessons learned from the prodebt research project on planning technical debt strategically**. pages 523– 534.
S17	Bomfim, M. and Santos, V. (2017). **Strategies for reducing technical debt in agile teams**. pages 60–71
S18	Martini, A. and Bosch, J. (2017b). **On the interest of architectural technical debt: Uncovering the contagious debt phenomenon**. Journal of Software: Evolution and Process, 29:e1877.
S19	de Assunçãao, T. R., Rodrigues, I., Venson, E., Figueredo, R. M. d. C., and De Sousa, T. L. (2015). **Technical debt management in the brazilian federal administration**. In 2015 6th Brazilian Workshop on Agile Methods (WBMA), pages 6–9.
S20	Caires, V., Rios, N., Holvitie, J., Lepp anen, V., Mendonc a, M., and Spínola, R. (2018). **Investigating the effects of agile practices and processes on technical debt - The viewpoint of the brazilian software industry**. pages 506–559.
S21	Rios, N., Spínola, R. O., Mendonça, M., and Seaman, C. (2018). **The most common causes and effects of technical debt: First results from a global family of industrial surveys**. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '18, New York, NY, USA. Association for Computing Machinery
S22	Holvitie, J., Licorish, S., Spínola, R., Hyrynsalmi, S., MacDonell, S., Mendes, T., Buchan, J., and Leppanen, V. (2017). **Technical debt and agile software development practices and processes: An industry practitioner survey**. Information and Software Technology, in press

Continued on next page

Table 1 – *Continued from previous page*

Id	Reference
S23	Martini, A., Besker, T., and Bosch, J. (2018). **Technical debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations**. Science of Computer Programming, 163:42–61
S24	Martini, A., Sikander, E., and Madlani, N. (2017). **A semi-automated framework for the identification and estimation of architectural technical debt: A comparative case-study on the modularization of a software component**. Information and Software Technology, 93.
S25	Toledo, S., Martini, A., Przybyszewska, A., and Sjøberg, D. (2019). **Architectural technical debt in microservices: A case study in a large company**.
S26	Rios, N., Mendonça, M., Seaman, C., and Spínola, R. (2019). **Causes and effects of the presence of technical debt in agile software projects**
S27	Martini, A., Stray, V., and Moe, N. (2019). **Technical-, social- and process debt in large-scale agile: An exploratory case-study**, pages 112–119.
S28	Martini, A. and Bosch, J. (2020). **Architectural technical debt in embedded systems**, pages 77–103.
S29	Rios, N., Spínola, R., Mendonça, M., and Seaman, C. (2020). **The practitioners' point of view on the concept of technical debt and its causes and consequences: A design for a global family of industrial surveys and its first results from brazil**. Empirical Software Engineering, 25
S30	Freire, S., Rios, N., Pérez, B., Castellanos, C., Correal, D., Ramac, R., Mandic, V., Tausan, N., Pacheco, A., López, G., Mendonça, M., Izurieta, C., Falessi, D., Seaman, C., and Spínola, R. (2021). **Pitfalls and solutions for technical debt management in agile software projects**. IEEE Software, 38(6):42–49.
S31	Malakuti, S. and Heuschkel, J. (2021). **The need for holistic technical debt management across the value stream: Lessons learnt and open challenges**. In 2021 IEEE/ACM International Conference on Technical Debt (TechDebt), pages 109–113.
S32	Bonfim, V. D., Benitti, F. B. V. (2022). **Requirements debt: Causes, consequences, and mitigating practices**. In Seke (pp. 13–18).
S33	Soares, G., Freire, S., Rios, N., Pérez, B., Castellanos, C., Correal, D., Mendonçaa, M., Izurieta, C., Seaman, C., and Spínola, R. (2022). **Investigating how agile software practitioners repay technical debt in software projects**. In Anais do XXI Simpósio Brasileiro de Qualidade de Software, pages 120–129, Porto Alegre, RS, Brasil. SBC.
S34	Wiese, M., Rachow, P., Riebisch, M., and Schwarze, J. (2022). **Preventing technical debt with the tap framework for technical debt aware management**. Information and Software Technology, 148:106926.
S35	Stochel, M. G., Wawrowski, M. R., and Chołda, P. (2022b). **Technical debt prioritization in telecommunication applications: Why the actual refactoring deviates from the plan and how to remediate it? Case study in the covid era**. Applied Sciences, 12(22).
S36	Eldh, S. (2022). **On technical debt in software testing - Observations from the industry**. Berlin, Heidelberg. Springer-Verlag.
S37	Stochel, M. G., Chołda, P., and Wawrowski, M. R. (2022a). **Adopting devops paradigm in technical debt prioritization and mitigation**. In 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pages 306–313
S38	Jamil, M. and Nour, M. (2022). **Managing software testing technical debt using evolutionary algorithms**. Computers, Materials and Continua, 73:735–747.
S39	Bonet Faus, J., Le Masson, P., Pelissier, U., Jibet, N., Bordas, A., and Pajot, S. (2023). **Design methods for diagnosing and locating entangled technical debt in devops frameworks**. Proceedings of the Design Society, 3:1267–1276.
S40	Aldaeej, A., Nguyen Duc, A., and Gupta, V. (2023). **A lean approach of managing technical debt in agile software projects – A proposal and empirical evaluation**, pages 67–76.
S41	Doshi, M. and Virparia, P. (2023). **Agile development methodology for software re- engineering**. In Goar, V., Kuri, M., Kumar, R., and Senjyu, T., editors, Advances in Information Communication Technology and Computing, pages 401–409, Singapore. Springer Nature Singapore.

3.1. Overview of selected studies

Figure 2 shows the distribution of studies according to publication year, application domain, venue type and study type. The first selected study was published in 2013, indicating a more recent interest in this field. This could be evidence that TD in CSE is a relatively new research topic. Also, analyzing the distribution per year, there is no concentration of publications in a specific year.

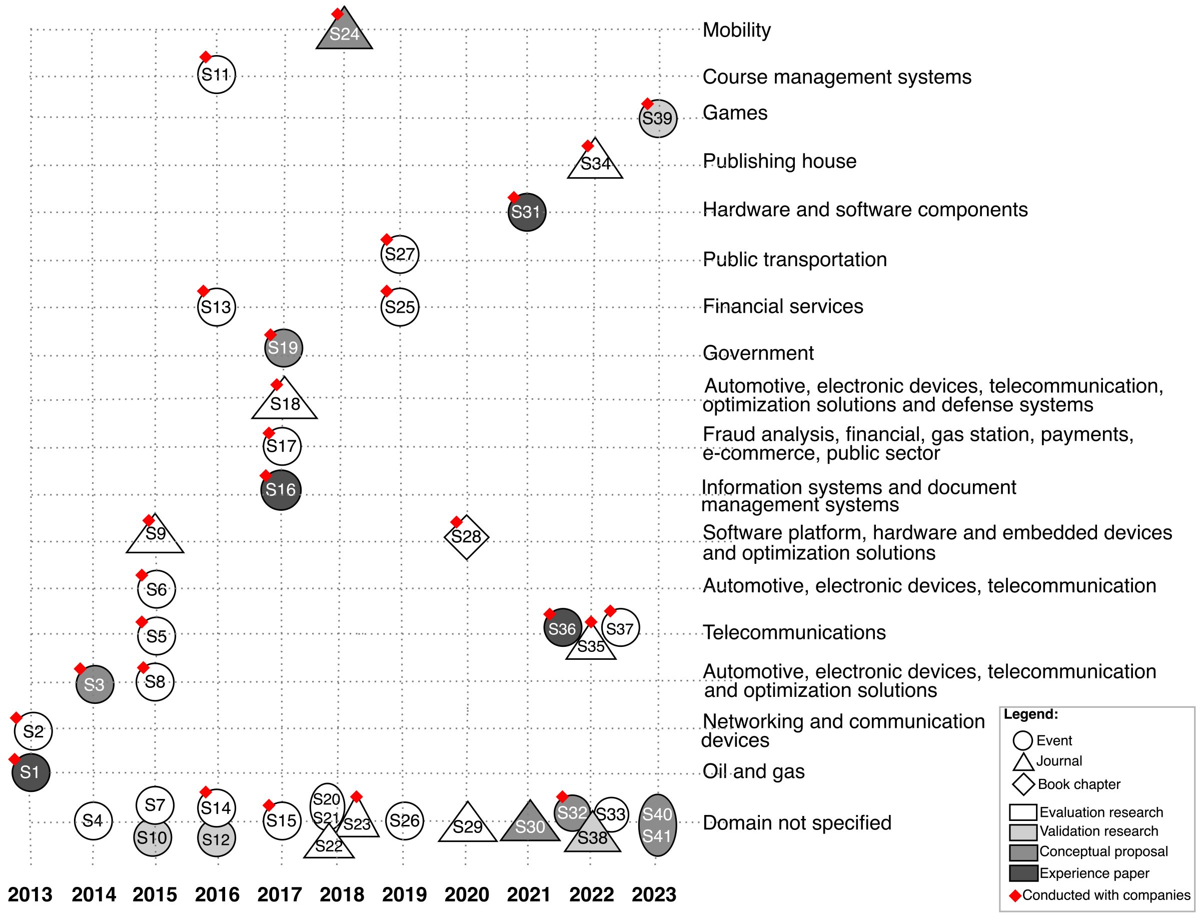

Figure 2. Overview of studies.

The majority of the studies (30) are published in events, while 10 were published in journals and only one as a book chapter. All publications are peer-reviewed, while in general, journal articles report more robust and complex studies. The only study published as a book chapter discusses about TD in CSE applied in the context of embedded systems, which is not discussed in other studies. Altogether, the combined observations may indicate that the research on TD in CSE is still maturing, with several open topics for investigation.

To analyze the study type, we used the classification proposed by [Wieringa et al. 2006]. In summary, studies classified as *evaluation research* are the most mature since they are based on empirical data and seek objective evidence about the results of an intervention, followed by studies classified as *validation research*. Studies classified as *solution proposal*, *conceptual proposal*, *experience paper*, or *opinion paper* correspond to those with less maturity; however, they contribute to introducing new ideas and emerging results. In the selected studies, 30/41 were classified as evaluation or validation research, which shows that this topic have been investigated mostly using empirical data and practical evidence (e.g., S8). The other 11/41 were classified as conceptual proposal or experience paper and are studies conducted generally in partnership with the industry to report a specific situation or phenomena (e.g., S24). Thus, these findings suggest that this topic has high relevance on the industry and is aligned with the interest of different companies that use one or more of CSE practices.

Regarding the application domains or types of systems, this information is directly extracted from the studies, which are mostly not specific about the application domain. This is mainly because these studies were conducted with practitioners but were not validated in a specific company, bringing a wider vision to the objective of the study instead of being limited to a specific context. The studies that specify a domain are conducted in a diverse set of areas, but many can be seen as critical, such as telecommunication (e.g., S5), government (e.g., S19) and finance (e.g., S25). Hence, this could show that this topic is being investigated with these kind of domains for having high relevance and impact in the industry. It also shows that this topic can be evaluated in different kinds of domains while maintaining its value.

Evaluating the studies that were conducted with the industry, it was mentioned that most studies are evaluation or validation research and this is endorsed by the fact that 27 studies were guided using data directly from different companies. There are different kinds of validation from these companies: some studies use the company as a case study to apply the research and collect the results (e.g., 35), while others gather results from a settled process in a company (e.g., S1), some others apply surveys or conduct interviews with practitioners to get answers and insights (e.g., S22). Case studies are the most used research method in studies that are conducted with companies and most of them are conducted in European companies. As a consequence, these findings validate that this topic has been mainly conducted in the industry. They also suggest that industry is also open to sharing their experiences to improve the state of the art and the state of the practice. Furthermore, there are two types of studies conducted in industrial contexts: the ones that are conducted by researchers who have some position in the company where the study is placed and the ones that are conducted by researchers with no relation to the context where the study takes place.

Finally, it is worth mentioning that all selected studies apply one or more of the known CSE practices, like agile development, CI, and continuous delivery, and analyze the TD that might be incurred in these practices. However, none of the selected studies discuss approaches to managing TD as a whole or how to characterize TD in CSE environments. Thus, a holistic approach to TD management in these environments is missing.

3.2. RQ - Which of the CSE activities address TD?

Figure 3, adapted from [Fitzgerald and Stol 2017], synthesizes the types of TD that are addressed or uncovered by the selected studies. The types are mapped to the flow of CSE as proposed by [Fitzgerald and Stol 2017].

The types of debt that we use are based on the classification of [Li et al. 2015]. Out of the studies, 19/41 do not mention any specific debt type related to the study,

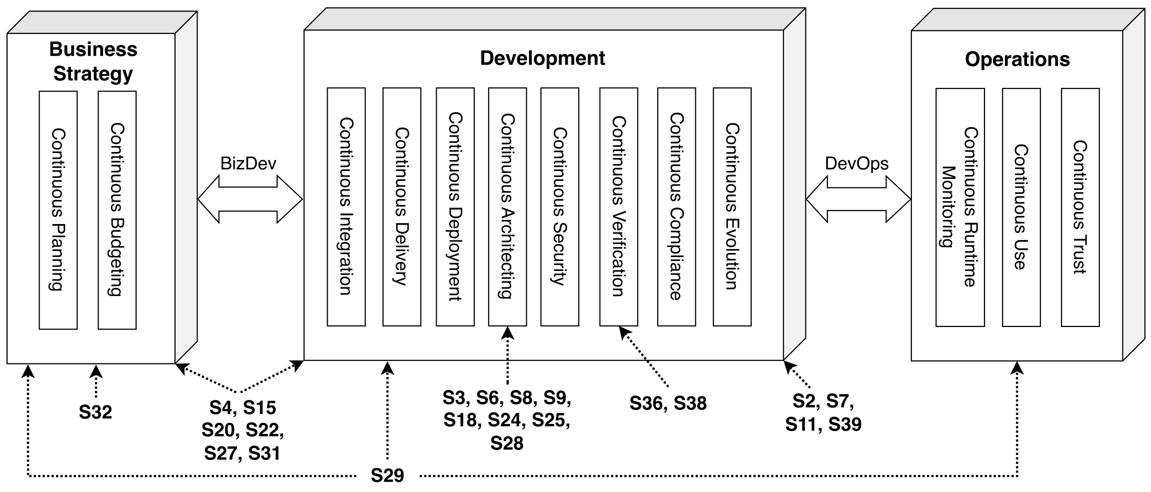

Figure 3. CSE activities in which TD was addressed (Based on [Fitzgerald and Stol 2017]).

focusing on other aspects of TD, like debt management, methods and approaches for TD. The rest of the studies (22/41) are specific about the debt type that is addressed or uncovered by the study; some of them mention more than one type and one of the studies mentions all of them and points out others not specified by [Li et al. 2015].

The studies mention these types of debt: architecture debt, build debt, code debt, defect debt, design debt, documentation debt, infrastructure debt, process debt, requirement debt, service debt, test debt, usability debt, versioning debt, but the CSE cycle is more generic and each step might encompass multiple activities of software engineering, so the studies in Figure 3 are connected to the step where the type of debt mentioned by the study might occur:

- S3, S6, S8, S9, S18, S24, S25, S28 explicitly handle architecture debt and the great number of studies specifically dealing with this type of debt show its importance;
- S36 and S38 deal with test debt and these studies discuss contemporary topics in continuous testing, like automation, TDD and test scenarios management;
- S32 is related to business strategy, but addresses requirements debt and this study mentions which facts and causes might happen in projects related to requirements engineering;
- S2, S7, S11 and S39 are associated to Development activities. S2 mention architecture, automation and infrastructure debt. S7 and S11 refers to documentation debt. S39 comments on design debt;
- S29 is connected to Business Strategy, Development and Operations because this study is related to different kinds of debt in the whole cycle: architecture debt, build debt, code debt, defect debt, design debt, documentation debt, infrastructure debt, process debt, requirement debt, service debt, test debt, usability debt and versioning debt; and
- S4, S15, S20, S22, S27, S31 are linked with both Business Strategy and Development because these studies mention types of debt that happen in both.

S4, S20 and S22 debates requirements, design, implementation and test debt. S15 touches upon architecture, design, documentation and test debt. S27 brings up architecture, infrastructure, documentation and test debt. S31 brings up architecture, requirements, build, code and test debt.

The most cited type of debt is architecture, followed by requirements, documentation and test debt, in no specific order. With regard to the CSE cycle, Business Strategy and Development activities are the ones with the most citation from the studies, while only one study addresses debt in Operations. The steps with the most citations are known for having high impact in the cycle of engineering a software [Singh and Gautam 2016] and all of them relate to quality. Therefore, there is an indication that TD has been more broadly studied in some steps of the CSE cycle because they usually interfere in main aspects of the quality of the software than in the rest of the CSE cycle, but more on this should be studied.

There are some activities in the CSE cycle that are not related to any kind of debt mentioned in the studies. Continuous budgeting, continuous compliance, and continuous trust are regular activities in some companies, but not directly related to the development cycle, so this might be the reason why there is no mention for a debt related to these activities. CI, continuous delivery, CD, and continuous security are widely adopted practices in the industry, but no study directly addresses TD in these activities. Continuous evolution activities might encompass TD management because it is usually the phase where the software is maturing and the issues related to it are found, so this might be the reason why the studies do not mention this activity. Continuous monitoring is another activity usually adopted in modern software development, specially in microservices, but no study mention any kind of debt related to it.

The studies that do not specify any kind of debt usually use some of the CSE practices, and apply some method for TD management, bringing light to a more generic approach. For instance, S19 proposes a strategy to check the quality of the code based on the TD volume, making TD identification and measurement in four stages: definition of violations, attribution of value to the pointers, measurement of non-conformances, and obtention of the results. Although studies like S19 are the minority of the selected studies, they indicate that TD might or might not be analyzed being related to a specific activity in the CSE cycle.

4. Discussion

The first study that was selected in this SMS was published in 2013, and this fact indicates that the interest on the intersection between TD and CSE practices is relatively new. Besides this, in the decade of 2010, the adoption of and the interest in CSE practices applied to software projects grew when compared to the previous decade in several places, like Fontana et al. [2022] exemplifies in Brazil, which might be one of the factors that drew attention to the discussion of TD in projects that use these practices. Thus, the combination of the growth in the adoption of CSE practices and the concerns related to software quality could have been the origin of this field of study.

While CSE activities have proved to be beneficial to software projects and business, the relationship between TD and CSE activities have not been fully addressed in previous studies. Our study shows, for instance in Figure 3, that some CSE activities have no TD study associated, which indicates the need for more research. As mentioned before, some of these uninvestigated activities (such as CI, continuous delivery, and continuous monitoring) have great impact in modern software development, and some of the studies address types of debt that are not in the original figure proposed by [Fitzgerald and Stol 2017].

Another finding is that industry is highly involved in studies addressing TD and CSE, as shown in Figure 2. Specifically, most of the selected studies (27/41) were either conducted by practitioners or used data collected from industrial settings (e.g.,companies or practitioners). To some extent, the growing interest on TD in CSE can indicate that bad practices (from which TD is derived) in CSE can have a high negative impact in the companies.

Finally, some application domains that normally adopt CSE activities, e.g., Industry 4.0 and Information Systems, have few or no studies addressing TD. This indicates that this topic has been researched in a varied range of domains, showing the industry is not focused on a single domain when addressing TD in CSE. It also shows that there are still open issues to be discussed in widely used types of systems. As an instance, S16 explores a prediction model in TD management in information systems and document management solutions and highlight that the main lessons learned were regarding data quality and evaluation, identification of key drivers of TD and credibility of prediction model for TD. Just as S16, when addressing TD in CSE in these types of systems or domains, some particularities could be found and used as a foundation for the evolution of the field.

4.1. Future perspectives

Considering the studies found in this SMS, it can be observed there are still diverse open issues to be addressed by future research and advance the state of the art in TD in CSE activities. Some of the main open issues are:

- Focus on CSE activities: Although several studies have addressed TD in CSE activities, some activities are still open for discovery and discussion of how TD is characterized and managed. These activities are CI, continuous delivery, CD, continuous security, and continuous monitoring. For future work, studies could specifically focus on these activities and verify which approaches or techniques are feasible for managing TD;
- TD management in the CSE as a whole: There is no study analyzing or evaluating TD management approaches or techniques destined for CSE as a whole or investigating its particularities. Hence, future works could present unified approaches, models, or techniques to manage TD considering all CSE activities, as well as applying them in practice. Besides this, we foresee that machine learning algorithms and supporting tools could be proposed to help uncover and manage TD in CSE activities; and
- TD in domains that adopt CSE: Some domains have usually applied CSE activities, e.g., Industry 4.0 and information systems, but few studies have addressed TD in systems of these domains. Hence, it is relevant to investigate

how TD could occur in these systems and how they could be managed in these domains' projects.

4.2. Threats to validity

In this section, we present and discuss the threats that could have potentially affected the validity of our SMS. There is also a discussion of the measures we took to mitigate these threats, considering the guidelines proposed by [Ampatzoglou et al. 2019], that are classified as study selection validity, data validity, and research validity, as presented below.

- **Study selection.** One of the most relevant threats to study selection is the selection of non-optimal digital libraries. In this study, we decided to use Scopus as a single database because it is highly relevant for Software Engineering research and includes a great number of publishers. Another threat is the ineffective search string and, to mitigate this threat, the search string was crafted and evaluated with experts in secondary studies and software engineering, and it was also piloted in the Scopus database. Another threat is the possibility that some relevant studies were omitted or some irrelevant ones were included and this threat was mitigated by having an SMS protocol with well-defined inclusion and exclusion criteria.
- **Data validity** Unverified data extraction and author bias are some of the most critical threats to the validity of the data, as they could lead to unreliable results and study conclusions. To mitigate these threats, we discussed all data items that could be extracted from the studies considering their meaning and relevance. Also, in the extraction phase, the data that the the first author extracted was revised by the second author.
- **Research validity** Replicability is one of the main concerns regarding our study, since analysing and synthesizing qualitative data was required to answer the proposed research question. To address this, we defined a detailed protocol based on the literature guidelines for SMS [Kitchenham et al. 2015, Petersen et al. 2008].

5. Conclusions

When using Agile processes, large software companies are making constant attempts to deliver software to the customer considering its value and its impact to the market. Agile and CSE have proved to be profitable to the industry, but these initiatives should not lead to the occurrence of bad decisions in development that have negative consequences in a long-term, named TD.

This study presents the state-of-the-art of how TD has been addressed in CSE activities. For this, we conducted an SMS and evaluated 41 studies. The findings indicate that this field of study is considerably new and has emerged in 2013, that TD has been addressed in CSE mainly with an industry interest or involvement and that, although some CSE activities have studies addressing TD, critical ones (e.g., CI, CD, continuous monitoring) are not addressed and, surprinsingly, no study address TD in CSE a holistic way. The future perspectives for the field are exciting, so studies could be conducted analyzing TD on not yet addressed CSE activities, evaluating approaches or techniques for TD management and using automation and algorithms to help manage TD.

Acknowledgment

This study was financed in part by National Council for Scientific and Technological (CNPq) (Grant No.: 313245/2021-5), São Paulo Research Foundation (FAPESP) (Grant No.: 2023/00488-5), and Coordination for the Improvement of Higher Education Personnel (Capes).

References

Alves, N. S., Mendes, T. S., de Mendonça, M. G., Spínola, R. O., Shull, F., and Seaman, C. (2016). Identification and management of technical debt: A systematic mapping study. *Information and Software Technology*, 70:100–121.

Ampatzoglou, A., Bibi, S., Avgeriou, P., Verbeek, M., and Chatzigeorgiou, A. (2019). Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. *Information and Software Technology*, 106:201–230.

Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, 32(2):50–54.

Codabux, Z. and Williams, B. (2013). Managing technical debt: An industrial case study. In *4th International Workshop on Managing Technical Debt (MTD)*, page 8 – 15.

Dingsøyr, T., Nerur, S., Balijepally, V., and Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85(6):1213–1221.

Fitzgerald, B. and Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189.

Fontana, R. M., Wojciechowski, J., Montaño, R. R., Marczak, S., Reinehr, S., and Malucelli, A. (2022). A countrywide descriptive survey of agile software development in brazil. In Stray, V., Stol, K.-J., Paasivaara, M., and Kruchten, P., editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 185–202, Cham. Springer International Publishing.

Kitchenham, B. A., Budgen, D., and Brereton, P. (2015). *Evidence-Based Software Engineering and Systematic Reviews*. Chapman & Hall/CRC.

Klotins, E. and Gorschek, T. (2022). Continuous software engineering in the wild. In Mendez, D., Wimmer, M., Winkler, D., Biffl, S., and Bergsmann, J., editors, *Software Quality: The Next Big Thing in Software Engineering and Quality*, pages 3–12, Cham. Springer International Publishing.

Kruchten, P., Nord, R. L., and Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *IEEE Software*, 29(6):18–21.

Li, Z., Avgeriou, P., and Liang, P. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101:193–220.

Papatheocharous, E. and Andreou, A. (2013). Evidence of agile adoption in software organizations: An empirical survey. In *European Conference on Software Process Improvement (EuroSPI)*, volume 364, pages 237–246.

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008). Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, page 68–77.

Seaman, C., Guo, Y., Zazworka, N., Shull, F., Izurieta, C., Cai, Y., and Vetro, A. (2012). Using technical debt data in decision making: Potential decision approaches. In *3rd International Workshop on Managing Technical Debt (MTD)*, pages 45–48.

Singh, B. and Gautam, S. (2016). The impact of software development process on software quality: A review. In *8th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 666–672.

Vijayasarathy, L. and Turk, D. (2012). Drivers of agile software development use: Dialectic interplay between benefits and hindrances. *Information and Software Technology*, 54(2):137–148.

Wieringa, R., Maiden, N., Mead, N., and Rolland, C. (2006). Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering*, 11(1):102–107.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, New York, NY, USA. Association for Computing Machinery.