Indonesian Journal of Computer Science
ISSN 2549-7286 (*online*)
Jln. Khatib Sulaiman Dalam No. 1, Padang, Indonesia
Website: ijcs.stmikindonesia.ac.id | E-mail: ijcs@stmikindonesia.ac.id

Lean and Agile Software Development for Managing Technical Debt on A Large-scale Software: A Systematic Literature Review

Surya Seven Y. Simangunsong[1], Teguh Raharjo[2], Anita Nur Fitriani[3]
simangunsong.seven@gmail.com, teguhr2000@gmail.com, nurfitriani.anita@gmail.com
[1,2,3]Faculty of Computer Science, Master of Information Technology, University of Indonesia, Jakarta

Article Information

Submitted : 17 Dec 2023
Reviewed: 22 Dec 2023
Accepted : 30 Dec 2023

Keywords

Lean, Agile, Technical Debt, Technical Debt Management Framework, Systematic Literature Review

Abstract

Agile methodologies are employed by software development teams for collaboration and adapting to changing requirements. However, this flexibility may lead to technical debt (TD), causing potential bugs in the long term. Lean principles, focusing on waste elimination and continuous process improvement, can be applied to manage TD in agile software development. This research conducts a systematic literature review on using lean and agile methodologies for TD management. The review identifies 34 papers, categorizing TD types, pinpointing lean and agile principles, and aligning technical debt categories with suitable lean and agile principles. Additionally, three existing technical debt management frameworks are identified: the TAP framework, the LTD framework, and the CoDVA framework. The study concludes that integrating lean principles into agile software development assists organizations in effectively managing technical debt. Furthermore, the research offers insights into selecting the most suitable TD management framework based on an organization's needs and available resources.

Indonesian Journal of Computer Science
ISSN 2549-7286 (*online*)
Jln. Khatib Sulaiman Dalam No. 1, Padang, Indonesia
Website: ijcs.stmikindonesia.ac.id | E-mail: ijcs@stmikindonesia.ac.id

A. Introduction

Agile methodology is an approach to software development that emphasizes flexibility, team collaboration, and adaptation to changing requirements during the development process. One of the Agile methodology principles is the ability to respond to changes quickly and effectively. Agile methods have revolutionized the way software development is conducted, emphasizing active end-user involvement, tolerance for changes, and iterative product delivery [1].

In the realm of software industry practices, comprehensive planning and well-designed software architecture don't always accompany software development. This results in the emergence of technical debt, a term in software development denoting the adverse effects of developer choices and actions that prioritize short-term gains or speed over long-term quality. Technical debt occurs when developers opt for swift, straightforward, or less-than-optimal solutions to address immediate requirements without considering their repercussions on the entire system [2].

Like financial debt, technical debt carries inherent interest, which involves additional costs or adverse effects stemming from less-than-optimal solutions. If this accrued interest becomes excessively high, it has the potential to trigger disruptive events such as development crises. Therefore, managing technical debt is crucial to prevent long-term losses. The primary goal is to improve software quality, reduce maintenance costs and risks, ensure sustainability, and respond to changes quickly. By controlling technical debt and minimizing its negative effects, companies can avoid costly development crises and ensure long-term success in software development[3].

Agile allows teams to adapt quickly, but the rapid pace can lead to technical debt. However, not all debt needs immediate attention, as organizations may lack the necessary budgets and resources. This is especially true when developing large-scale software with limited resources. Handling technical debt in such cases should be based on assessing the risks and impacts on the software, as well as the existing business priorities [2]. Large-scale systems are typically characterized by high complexity, scalability required to handle growth and high demand, the ability to process large-scale data with good performance, and an architecture that enables efficient system changes and evolution [4].

Holvitie et al. [5] carried out an international survey aimed at classifying the impacts of introducing agile methodologies into the management of technical debt (TD). A study of 184 professionals in three countries found that practitioners understand TD but need to improve its implementation. TD is often identified in legacy systems, but it can be difficult to provide specific examples, which complicates its management. The agile practices and processes studied help reduce TD and approaches that validate and maintain the structure and clarity of implemented artifacts, such as coding standards and refactoring, have a positive impact on TD management. As a result, the parallels in TD instances indicate the practicality of instituting a systematic approach to TD management.

A study by Digkas et al. [6] investigated the relationship between using code from Stack Overflow and the accumulation of Technical Debt (TD) in software systems. Their findings suggest that reusing small amounts of code can increase TD, highlighting the importance of code quality assessment before integrating external code. In a separate study, Gama et al. [7] examined how Stack Overflow users identify TD issues in their projects. They

discovered that developers actively engage in discussions about TD identification, revealing 29 specific indicators for recognizing architecture-related, code-related, test-related, and infrastructure-related TD issues.

In a different research endeavor, Besker and their team explored the specific effects of Architectural Technical Debt (ATD) on software practitioners [8]. The outcomes of the study demonstrated the significant adverse impact of architectural ATD on the daily activities of software practitioners and its ramifications on various roles within the realm of software development. Moreover, the software's age was identified as an influencing factor in the extent of time wasted due to ATD. In a later investigation conducted by Besker and colleagues [8], the findings indicated that around 36% of the time allocated to development is lost because of Technical Debt (TD), particularly focusing on architectural and requirement related ATD. Additionally, the research indicated that software practitioners frequently introduce new TD issues because of existing ones.

Within the same study, Martini and Bosch [9] probed into the consequences of technical debt, reporting that TD items can be infectious, leading to the contamination of other segments within the system with the same problem, potentially resulting in nonlinear growth in interest.

Addressing technical debt is not cheap, so an efficient software development strategy is needed. This involves dealing with existing technical debt and prioritizing software quality in future development. One approach to achieving this, which focuses on eliminating waste and improving efficiency in the development process, is Lean Software Development [10]. A vital principle in Lean Software Development is preventing the buildup of technical debt. This means developers should prioritize quality, cleanliness, and continuity in code and software design. Following this principle helps the development team minimize the accumulation of technical debt, ensuring the system can evolve, be fixed, and improve efficiently [11].

The high cost of technical debt in large software projects is driving research into how lean and agile approaches can be used to manage it. This research will analyze the principles of both approaches and combine, adapt, and implement them in software development. It will also discuss the benefits and challenges teams face when developing large-scale software with limited resources, and how to prioritize which technical debts to address. The goal is to provide insights and practical guidance for companies on how to successfully manage technical debt.

B. Research Method

The research method used in this study uses Systematic Literature review (SLR). Systematic Literature Review (SLR) is a systematic and structured research method used to identify, assess, and interpret all available research evidence to provide the answers to research questions. Through the Systematic Literature Review, researchers can identify gaps in existing research, gather existing evidence, and produce an objective and transparent summary of a specific topic [20]. The research questions for this research are:

RQ1: What are the types of Technical Debt?

RQ2: How to use the Lean Software Development and Agile approaches in managing Technical Debt on large-scale software?

Indonesian Journal of Computer Science
ISSN 2549-7286 (*online*)
Jln. Khatib Sulaiman Dalam No. 1, Padang, Indonesia
Website: ijcs.stmikindonesia.ac.id | E-mail: ijcs@stmikindonesia.ac.id

RQ3: According to Lean and Agile principles, which framework can be used for managing technical debt?

A) Literature Study

This section will explain some terms used in this research. In general, this section elaborates on technical debt, agile approach, lean software development, lean and agile.

The concept of code debt (technical debt) was first introduced by Ward Cunningham in 1992. Cunningham used the analogy of debt to convey the idea that when development teams take shortcuts or sacrifice technical quality in their development process, they 'owe' the system and will incur additional costs later on [12]. Technical debt consists of three main concepts: Debt: The amount of effort required to address existing problems; Interest: The additional cost needed to repay the technical debt; Principal: The technical cost incurred because of decisions made to resolve issues stemming from technical debt, also known as the cost of refactoring [12].

Technical debt (TD) is categorized into two types: intentional and unintentional technical debt. Unintentional technical debt can be caused by selecting simple or not optimal solutions to meet immediate needs, without considering the impact on the system, such as implementing software with poor code quality. Intentional technical debt arises when the development team chooses or makes shortcuts in developing a solution. Both types of technical debt can result in similar problems, such as slowing down future development and deteriorating code quality. Therefore, it is crucial for the development team to understand and effectively manage technical debt [13].

In Figure 1, the diagram illustrates the progression of technical debt over time when left unattended. As the technical debt increases, so does the associated interest. In the absence of technical debt, maintenance costs stay consistently at an optimal level [14].

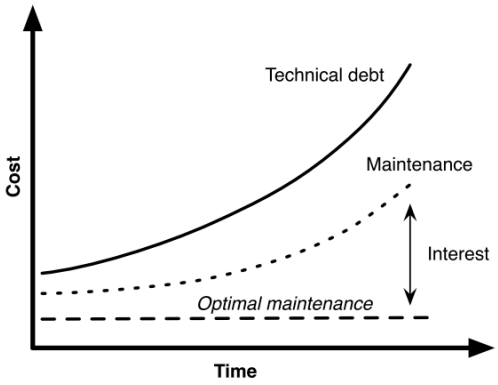

Figure 1. Technical debt and its associated costs accumulate over time [14]

The Agile approach is a flexible and collaborative approach to software development. The use of Agile methodologies aims to deliver business value and enable quick adaptation to changes throughout the development process. The Agile Manifesto, published in 2001, provides the foundational framework for this approach. The Agile Manifesto emphasizes the importance of collaboration, responsiveness to change, delivering functional software, and

Indonesian Journal of Computer Science
ISSN 2549-7286 (*online*)
Jln. Khatib Sulaiman Dalam No. 1, Padang, Indonesia
Website: ijcs.stmikindonesia.ac.id | E-mail: ijcs@stmikindonesia.ac.id

focusing on individuals and interactions in software development. The Agile Manifesto has 4 core values and 12 principles that guide the Agile approach in software development projects. These values prioritize individuals and interactions over processes and tools, highlight the significance of working software over extensive documentation, encourage collaboration with customers over contractual negotiations, and endorse adaptability to respond to changes swiftly [15]. The Agile Manifesto can be seen in Figure 2.

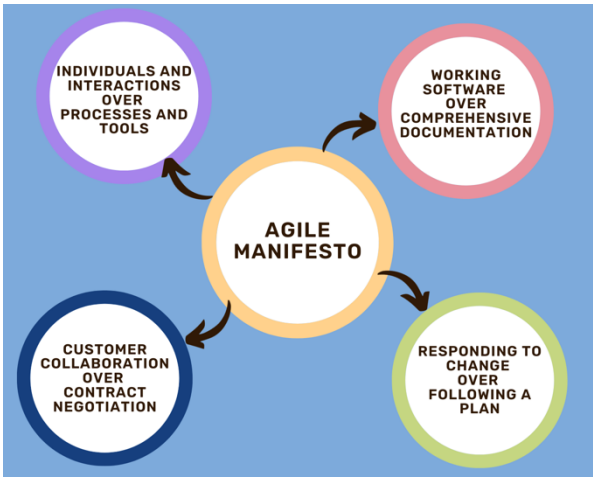

Figure 2. Agile Manifesto [15]

Agile promotes avoiding a large build-up of technical debt. Through iterative cycles and a focus on delivering functional software, teams aim to consistently uphold the quality and sustainability of the software. Regularly addressing technical debt helps teams prevent substantial accumulations and lowers the risk of expensive changes in the future [16]. Agile software development is susceptible to technical debt due to its focus on quick project delivery. Prioritizing immediate functionality in short iterations may lead to overlooking long-term software quality. Time constraints and pressure to meet deadlines can lead to decisions that compromise code quality, thorough testing, documentation, or architectural considerations. [17].

Lean first became known in the post-World War II era in the 1940s. This approach emerged as part of the industrial renaissance and was heavily influenced by the Toyota Production System (TPS). At that time, Japan faced significant resource limitations, including limited capital, raw materials, and skilled labor. In response to these challenges, Toyota sought ways to improve efficiency and eliminate waste in its manufacturing processes. Taiichi Ohno, an engineer at Toyota, played a key role in developing the system that later became known as Lean. However, the term 'Lean' was not used until researchers from MIT mentioned that Lean is about ‘doing more with less’ by producing ‘the right things, at the right time and in the right place’ [18]. In **Error! Reference source not found.**, five principles in the Lean concept, which are [11]:

Indonesian Journal of Computer Science
ISSN 2549-7286 (*online*)
Jln. Khatib Sulaiman Dalam No. 1, Padang, Indonesia
Website: ijcs.stmikindonesia.ac.id | E-mail: ijcs@stmikindonesia.ac.id

1. ***Value***: In Lean, the first principle centers on recognizing value from the customer's viewpoint. This entails understanding what customers consider valuable, their needs, and how a product or service can deliver significant benefits. By grasping customer preferences, organizations can concentrate on developing products or services that meet those needs.
2. ***Value Stream***: The second principle involves creating a seamless value stream in the business process. This encompasses the steps needed to transform raw materials into finished products or deliver services. The goal is to identify and eliminate waste in the value stream, such as waiting time, rework, unnecessary transportation, and excess inventory.
3. ***Flow***: The third principle is about establishing a smooth flow in the value stream. This includes optimizing the workflow to ensure tasks are performed without hindrances.
4. ***Pull***: This principle is the concept of a responsive "pull" system based on customer demand. This means producing goods or services based on actual demand rather than on inventory.
5. ***Strive for Perfection***: The final principle highlights continuous improvement through Kaizen practices. The team is encouraged to continually look for ways to improve processes, quality, and efficiency, to make continuous improvements and deliver superior value to customers.

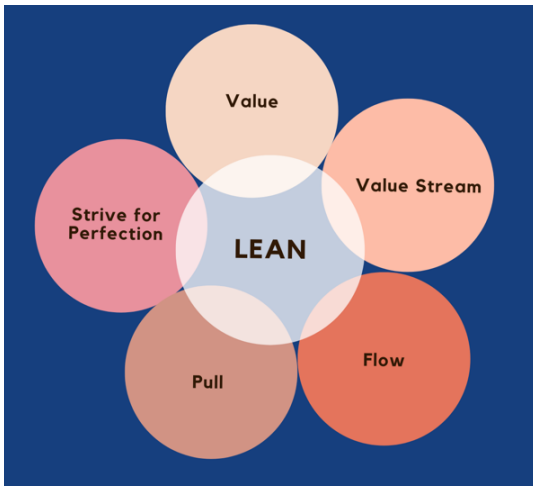

Figure 3. Lean principles [11]

When Agile focuses on flexibility and adaptability in dealing with changes to deliver customer satisfaction through iterations and feedback, Lean Software Development focuses more on eliminating waste in software development and improving efficiency.

Lean Software Development uses methods like Value Stream Mapping, Kanban, Just-in-Time (JIT), and Continuous Integration (CI) to cut waste and boost value. The main goal is to operate with minimal resource use, covering people, space, storage, tools, and time. While

prioritizing efficiency and minimizing resources, Lean may reduce team collaboration and adaptability to sudden changes. If significant shifts in requirements occur, Lean might struggle to adjust quickly. Combining Lean with Agile creates five values, which are [19]:

1. ***Customer Value***: Agile emphasizes gathering customer "user stories" that describe specific features and outcomes valued by the customer. It's crucial to discuss these stories in language that captures the customer's needs.
2. ***Continuous Improvement***: Sprints, lasting 1-2 weeks, aim to shorten the learning cycle and adjust plans based on outcomes and customer feedback. Agile avoids lengthy design and development cycles, adapting to changing project needs, aligning with Lean's focus on multiple iterations for significant improvement.
3. ***Flexibility***: Agile was designed to help organizations navigate constant change. Successful organizations are flexible, adapting quickly to changing customer needs, competition, and markets, with flexible organizational structures.
4. ***Respect***: The Agile manifesto prioritizes "Individuals and interactions over processes and tools." In high-performing teams, everyone is respected and trusted to complete their chosen task.
5. ***Flow***: Agile focuses on smart work for greater value with less effort, following Lean's flow principles. Each sprint completes user stories, delivering new features, and addressing documentation, testing, and user-interface requirements upfront for faster feedback and precise documentation. Sprints reduce traditional batching, akin to Lean's "continuous flow." Some Agile methods, like Kanban, draw from Lean's just-in-time concept, emphasizing efficiency in development.

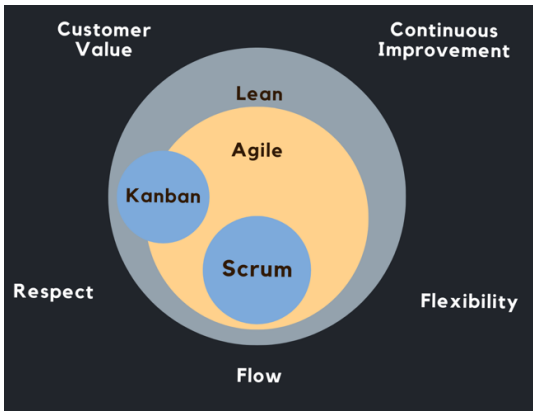

Figure 4. Combining Lean with Agile [19]

In Figure 4, the previous research has looked into combining Lean and Agile methods, but there's not much on how they integrate in managing technical debt for large-scale software. This study aims to find new and effective ways to handle technical debt in large-scale software systems by merging Lean Software Development and Agile approaches. The goal is

to help organizations identify, reduce, and prevent technical debt, leading to improved quality and sustainability in complex systems.

The results of this research can provide insights and practical guidance for organizations to effectively manage technical debt, optimize resource utilization, improve development efficiency, and produce software systems that are more maintainable and of high quality in the future.

B) Research Design

This study uses the Kitchenham method which consists of planning, implementation, and reporting stages. Each stage of the Systematic Literature Review (SLR) will be discussed in the following subsections.

1) Planning the SLR

In this stage, the researcher identifies the research objectives and research questions to be answered. Then, a search is conducted based on the predetermined research questions. The databases used for searching related research include ACM, IEEE Xplore, Google Scholar, Springer, and ScienceDirect. In searching for related research, the first set of keywords used in the search process is related to the approach, way, and advancement in Lean Software Development and Agile. These keywords are ("APPROACH" OR "WAY" OR "ADVANCE" OR "FRAMEROWK"). The second set of keywords is ("CHALLENGES" OR "OBSTACLES" OR "ISSUES"). The third set of keywords is ("LEAN" OR "AGILE" OR "SOFTWARE" OR "DEVELOPMENT"). The first and second sets of keywords are combined to form ("APPROACH" OR "WAY" OR "ADVANCE") AND ("CHALLENGES" OR "OBSTACLES" OR "ISSUES"). Then, these sets of keywords are further combined with "managing technical debt in large-scale software". Therefore, the final set of keywords used in the database search is ("APPROACH" OR "WAY" OR "ADVANCE") AND ("CHALLENGES" OR "OBSTACLES" OR "ISSUES") AND ("LEAN" OR "AGILE" OR "SOFTWARE" OR "DEVELOPMENT") AND ("TECHNICAL" OR "DEBT").

In the search results, inclusion (IN) and exclusion (EX) filters were applied to identify relevant papers. The IN1 criterion is the search timeframe, which is from 2019 to 2023. The IN2 criterion is the English language for writing, the IN3 criterion is the type of publication (journal and international conferences), the IN4 criterion is selecting the field of engineering and computer science, and the IN5 criterion is choosing publications that are in the final stage. The exclusion (EX) criterion, EX1, involves eliminating research papers with titles unrelated to Lean and Agile in project management or technical debt management. Additionally, elimination was performed based on abstracts that were not related to Lean and Agile in tech debt management.

2) Implementation of SLR

The paper selection process began with 700 articles identified through keywords and criteria (IN1, IN2, IN3, IN4, IN5). After applying the EX1 criterion, 200 articles remained. Further elimination based on irrelevant abstracts resulted in 19 relevant papers focused on Lean and Agile in tech debt management. The snowballing technique, exploring reference lists, added 5 more relevant papers. A Google Scholar search contributed 10 additional papers, bringing the total to 34 relevant papers. These papers were used for data extraction, and their synthesis addressed the research questions. See Figure 5 for the search process.

Indonesian Journal of Computer Science
ISSN 2549-7286 (*online*)
Jln. Khatib Sulaiman Dalam No. 1, Padang, Indonesia
Website: ijcs.stmikindonesia.ac.id | E-mail: ijcs@stmikindonesia.ac.id

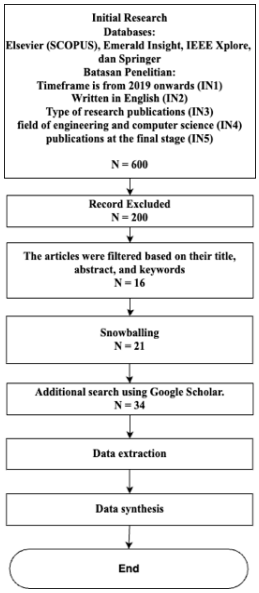

Figure 5. The procedure of this mapping study

3) Reporting the SLR

The study uses past research to answer formulated research questions. For RQ1, the researcher identifies types of technical debt based on studies by [13] and [20]. Addressing RQ2, the researcher starts with the foundational principles of Lean and Agile, discussing their role in software development, including the values in the Agile Manifesto, Lean Software Development, and waste in software development. This discussion leads to identifying practices for combining Lean and Agile, particularly in managing technical debt. The researcher then identifies frameworks for managing technical debt from past research and explains each. Finally, recommendations for user needs are provided. The reporting method for the Systematic Literature Review (SLR) is shown in Figure 6.

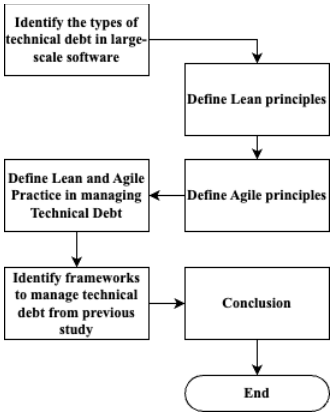

Figure 6. SLR reporting technique

Indonesian Journal of Computer Science
ISSN 2549-7286 (*online*)
Jln. Khatib Sulaiman Dalam No. 1, Padang, Indonesia
Website: ijcs.stmikindonesia.ac.id | E-mail: ijcs@stmikindonesia.ac.id

The data search process was conducted from 2019 to the end of October 2023. The results showed that 34 research papers were used as data for the study. Table 1 provides a distribution of the 34 research sources used. The largest number of high-quality research sources was obtained from IEEE Xplore. The distribution of studies by years can be seen in Figure 7.

Table 1. Result mapping online database

Database	Quantity	(%)
IEEE Xplore	11	32.35%
SpringerLink	5	14.71%
Google Scholar	8	23.53%
ScienceDirect	9	26.47%
ACM	1	2.94%
Total	34	100%

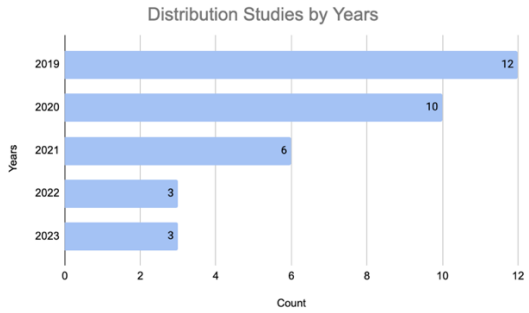

Figure 7. Distribution of research sources by publication year

C. Result and Discussion

This chapter will discuss the results of the conducted research. It includes the categorization of Technical Debt (TD), followed by a discussion on the Lean and Agile approaches in Software Development. Additionally, a mapping of the challenges faced in managing Technical Debt will be conducted.

A) Technical Debt Classification

To answer RQ1, a literature review was conducted on previous studies. In categorizing TD, the researchers carried out mapping, which can be seen in Table 2. Related literature on TD is conducted as a prerequisite for determining the approach in managing technical debt. Based on Table 2 most common types of software development technical debt are Architectural (Constructive) TD, Design-related TD, Test-related TD, Code TD and Requirement TD. Using the results of this classification, mapping will be carried out to determine solutions for each technical debt during implementation using Lean and Agile methodologies.

Table 2. Technical Debt classification

Indonesian Journal of Computer Science
ISSN 2549-7286 (*online*)
Jln. Khatib Sulaiman Dalam No. 1, Padang, Indonesia
Website: ijcs.stmikindonesia.ac.id | E-mail: ijcs@stmikindonesia.ac.id

No	Technical Debt Type	Technical Debt Subtype
1	Requirements TD	Incomplete Requirement [14], [21]
		Over-engineering [21], [22]
2	Architectural (Constructive) TD	Architecture smells [2], [23], [24]
		Violations of good architectural practices [23], [25]
		System-level structure quality issues [19], [20], [21]
3	Design-related TD	Code smells [28], [29]
		Complex classes or methods [29]
		Incompatible design[23], [29], [30]
4	Code TD	Low-quality code [28], [31]
		Duplicate code [28], [32]
		Complex code [28], [32]
5	Test-related TD	Code smells [28], [29]
		Complex classes or methods [29]
		Incompatible design[23], [29], [30]
6	Documentation TD	Low-quality code [28], [31]
		Duplicate code [28], [32]
		Complex code [28], [32]
7	Infrastructure TD	Old technology Issue [22]
		Lack of continuous Integration [33], [34]
		Poor release Planning [35]

B) Lean and Agile Practices in Managing Technical Debt

Based on previous research, it was found that Lean principles generally consist of 5 principles as shown in Table 3. Meanwhile, Agile principles consist of 8 principles, as can be seen in

Table 4.

Table 3. Lean principles

No	Lean Principles	Related Study
1	Define value from costumer perspective	[1], [10], [11], [15], [17], [19], [36]–[38]
2	Identify the value stream	
3	Make the flow	
4	Implement pull based production	
5	Strive for perfection continuously	

Indonesian Journal of Computer Science
ISSN 2549-7286 (*online*)
Jln. Khatib Sulaiman Dalam No. 1, Padang, Indonesia
Website: ijcs.stmikindonesia.ac.id | E-mail: ijcs@stmikindonesia.ac.id

Table 4. Agile principles

No	Agile Principles	Related Study
1	Customer satisfaction	[1], [10], [11], [15],
2	Embrace changing requirements:	[17], [19], [36]–[38]
3	Deliver working software frequently	
4	Collaborative approach	
5	Technical excellence	
6	Functional products	
7	Simplicity	
8	Self-organized teams	

By using Lean and Agile principles, mapping is done between types of technical debt and practices that can be implemented to prevent or address existing technical debt. This mapping can be seen in Table 5.

Table 5. Lean and Agile practices to handle TD

No	Technical Debt Type	Lean and Agile Practices
1	Requirements TD	- Involving stakeholders in requirement management [1] - Defining requirements based on the customer's perspective [36] - Prioritizing important technical debt that needs to be addressed immediately and less critical ones that can be worked on later [2]
2	Architectural TD	- evolutionary designs [38] - Regularly performing refactoring [11] - Involving every team member in architectural design decisions [1] - Conducting architecture evaluation through peer review, retrospectives, or other discussion forums [15]
3	Design TD	- Avoiding unnecessary complex designs [11] - Collaborating among team members with - diverse knowledge, such as architects, developers, and analysts [11] - Implementing an iterative approach to design [1], [15]
4	Code TD	- Performing code reviews [16] - Conducting refactoring [11] - Practicing pair programming [11]

No	Technical Debt Type	Lean and Agile Practices
5	Test TD	- Implementing the TDD (Test-Driven Development) approach [11] - Implementing automation testing [11], [39] - Applying Continuous Integration and Continuous Testing (CI/CT) [11] - Collaboration between developers and testers [11]
6	Documentation TD	- Documenting important and relevant information [40] - Updating documentation when there are changes [22] - Involving the team in testing and verifying documentation [11]
7	Infrastructure TD	- Implementing version control, continuous integration, and code review practices [14], [31] - Monitoring the performance, weaknesses, and security of the infrastructure [11]
8	Defect TD	- Implementing automation testing [41] - Organizing the defect backlog [11] - Conducting retrospectives to evaluate development [16] - Regular regression testing [41]

C) Framework for Technical Debt Management

Based on previous research, several frameworks have been identified for the management of technical debt in software development. These frameworks include the Technical Debt Aware Project (TAP) framework, the Less Technical Debt (LTD) framework, and the Continuous Debt Valuation Approach (CoDVA).

1) TAP Framework

The TAP Framework is a framework for managing technical debt/TD, and it was developed and established in an IT unit in early 2018. The focus of the IT unit that developed this framework is on systems that support advertising in print, online, and mobile media, a volatile and competitive market [42]. The TAP Framework flows can be seen in Figure 8.

Indonesian Journal of Computer Science
ISSN 2549-7286 (*online*)
Jln. Khatib Sulaiman Dalam No. 1, Padang, Indonesia
Website: ijcs.stmikindonesia.ac.id | E-mail: ijcs@stmikindonesia.ac.id

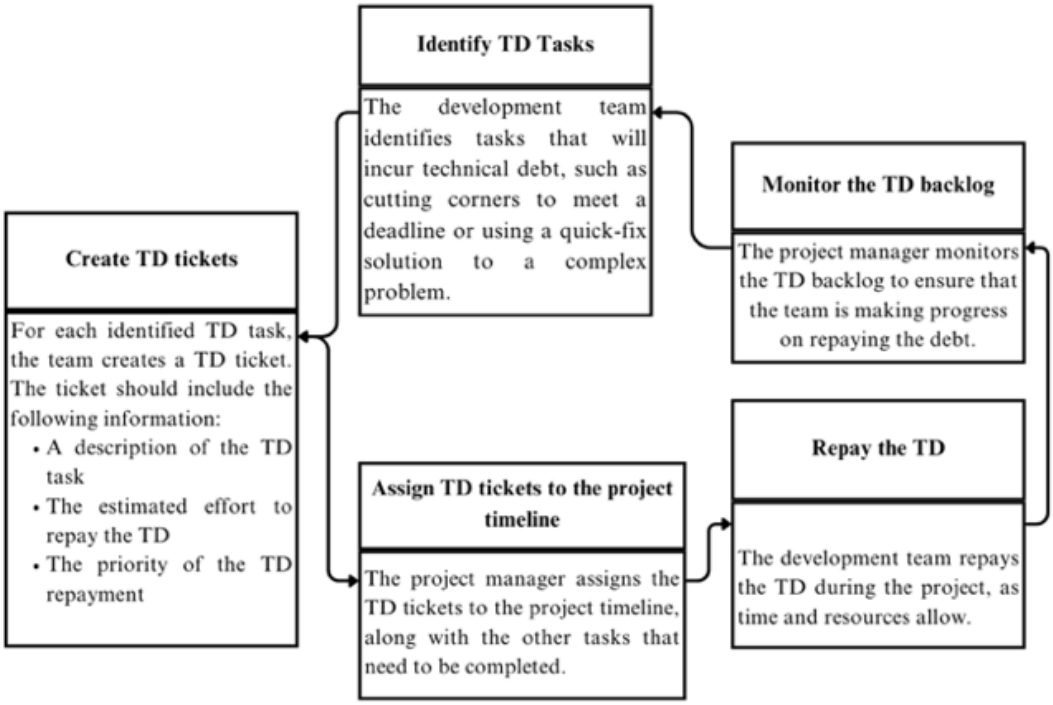

Figure 8. TAP framework [42]

2) LTD Framework

LTD is a framework that assists agile teams in managing technical debt (TD). LTD is a lightweight and flexible framework, making it adaptable to the needs of agile teams. Teams can choose to implement all or some of the LTD activities and can customize these activities to align with their existing workflow [43]. The LTD Framework flows can be seen in Figure 9.

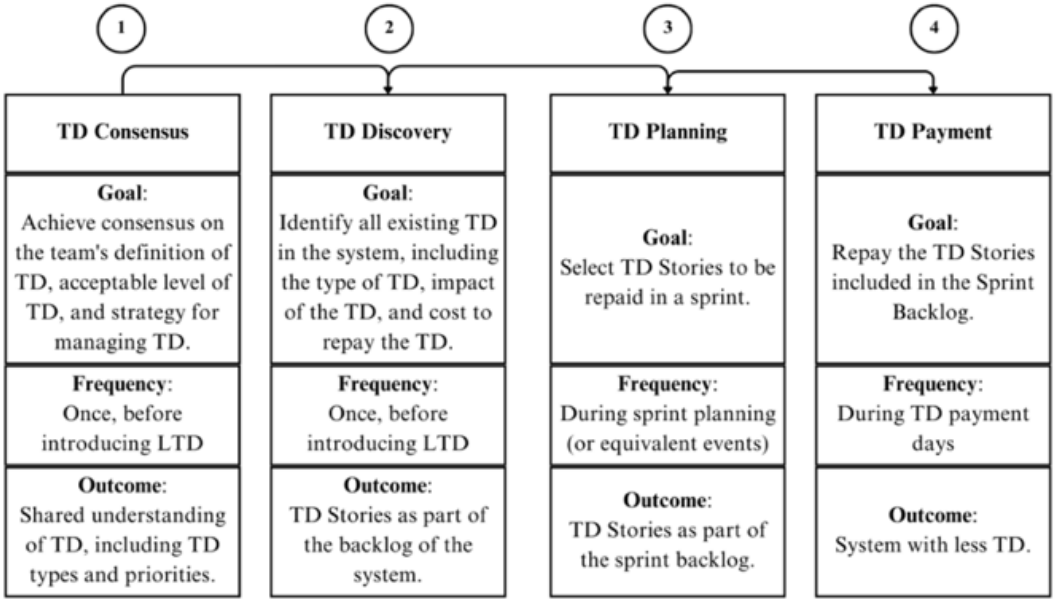

Figure 9. LTD Framework [43]

Indonesian Journal of Computer Science
ISSN 2549-7286 (*online*)
Jln. Khatib Sulaiman Dalam No. 1, Padang, Indonesia
Website: ijcs.stmikindonesia.ac.id | E-mail: ijcs@stmikindonesia.ac.id

3) CoDVA Framework

CoDVA is an approach for assessing and prioritizing technical debt (TD) relative to each other. CoDVA combines several factors, including refactoring size, expected benefits, business priorities, and inter-TDI dependencies [44]. The CoDVA Framework flows can be seen in Figure 10.

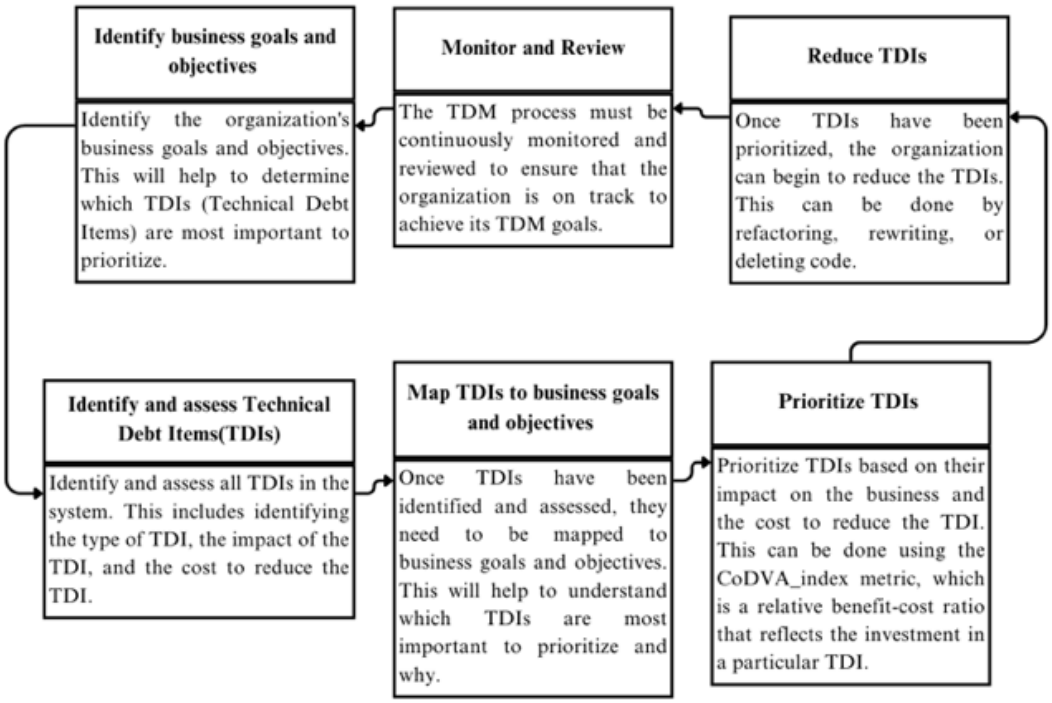

Figure 10. CoDVA Framework [44]

4) TAP, LTD, and CoDVA Framework

The TAP framework is a simple framework that can be adapted to the needs of agile teams. This framework is easy to implement and use for organizations that have small-scale software. [42]. The LTD framework is more comprehensive but guides all aspects of TDM, including identification, prioritization, and reduction of TD. Managing TD using this framework may be more complex, but it is more detailed than the TAP framework, so it is suitable for large-scale software [43].

The CoDVA framework is a quantitative framework that uses a metric called CoDVA_index to prioritize TD items. this framework. is the most complex framework compared to TAP and LTD. However, the advantage of this framework is that it identifies and prioritizes TD based on business objectives and can provide guidance on all aspects of TDM. Apart from that, this framework also includes TD monitoring and review activities. So, for software that is quite large and very strict with business objectives, this framework is suitable [44].

Comparison of frameworks for managing technical debt based on agile principles can be seen in Table 6.

Table 6. Comparison of frameworks for managing technical debt based on agile principles

Indonesian Journal of Computer Science
ISSN 2549-7286 (*online*)
Jln. Khatib Sulaiman Dalam No. 1, Padang, Indonesia
Website: ijcs.stmikindonesia.ac.id | E-mail: ijcs@stmikindonesia.ac.id

Practices	Categories	TAP [34]	CoDVA [36]	LTD[35]
Define value from costumer perspective [7]	Lean	×	✓	×
Identify the value [6]	Lean	✓	✓	✓
Technical Debt Prioritization [12]	Lean	✓	✓	✓
Pengelolaan biaya dan Sumber daya [5]	Lean	×	✓	✓
Fleksibilitas dan Adaptabilitas [38]	Agile	✓	×	✓
Dapat diintegrasikan dengan praktik *Scrum* [30]	Agile	✓	✓	✓

In the Table 6, in the define value point from the customer perspective, the CoDVA framework is used. In the identify value and TD prioritization points, all three frameworks have implemented their practices. The cost and resource management practices are applied by CoDVA and LTD. TAP and LTD have implemented practices for flexibility and adaptability. All three frameworks can be integrated with Scrum, making them suitable for organizations using that method.

D. Conclusion

This research aims to examine technical debt management practices and provide recommendations for a framework for managing technical debt. In this research, the types of technical debt in software development are categorized into 7 types. Next, the Lean and Agile principles were identified. Based on the identification of 7 types of technical debt, 5 Lean principles, and 8 Agile principles found in previous research, a mapping was carried out between each type of technical debt and related solutions or practices that can be implemented based on Lean and Agile Software Development principles. In this research, 3 frameworks were also identified that can be used to manage technical debt. This framework can be implemented to each organization's unique needs and characteristics. For organizations with large-scale software, the first option is recommended to use the CoDVA framework but will be more complex because it must calculate the CoDVA index first, and the second option uses the LTD framework. If an organization is small-scale software and looking for a lightweight and easy-to-use framework, then the TAP framework is a good choice.

This research is limited to research on SLR results, so it is possible that another relevant research is not included. There may be bias in the selection of literature because it depends on the researcher's abilities. Inaccuracies in data extraction and bias in data synthesis are also possible. During the data extraction process, inaccuracies can occur due to the reliance on a single researcher, which is subject to their knowledge and understanding. Bias in data synthesis may arise because not all papers provide detailed information on the data to be extracted, potentially reducing the accuracy of the synthesized data.

This study is primarily centered on the phases within the software development lifecycle that give rise to technical debt (TD). It involves the identification of appropriate strategies aligning with Lean and Agile principles to mitigate such TD. Subsequent research endeavors might delve into further classifications of TD and explore prevalent TD challenges faced

during the development of extensive software projects. Additionally, this study has the potential for expansion by integrating Lean principles with other methodologies like DevOps, Waterfall, Rapid, Prototype, and others. Researchers can explore each framework individually or assess potential synergies between these methodologies to develop innovative, optimized frameworks for TD management.

E. References

[1] M. Zorzetti, I. Signoretti, L. Salerno, S. Marczak, and R. Bastos, "Improving Agile Software Development using User-Centered Design and Lean Startup," *Inf Softw Technol*, vol. 141, Jan. 2022, doi: 10.1016/j.infsof.2021.106718.

[2] V. Lenarduzzi, T. Besker, D. Taibi, A. Martini, and F. A. Fontana, "Technical Debt Prioritization: State of the Art. A Systematic Literature Review," Apr. 2019, [Online]. Available: http://arxiv.org/abs/1904.12538

[3] M. Soliman, P. Avgeriou, and Y. Li, "Architectural design decisions that incur technical debt — An industrial case study," *Inf Softw Technol*, vol. 139, Nov. 2021, doi: 10.1016/j.infsof.2021.106669.

[4] T. Bures *et al.*, "Software Engineering for Smart Cyber-Physical Systems (SEsCPS 2018) - Workshop Report," *ACM SIGSOFT Software Engineering Notes*, vol. 44, no. 4, pp. 11–13, Dec. 2019, doi: 10.1145/3364452.3364465.

[5] J. Holvitie *et al.*, "Technical debt and agile software development practices and processes: An industry practitioner survey," *Inf Softw Technol*, vol. 96, pp. 141–160, 2018.

[6] G. Digkas, N. Nikolaidis, A. Ampatzoglou, and A. Chatzigeorgiou, "Reusing code from stackoverflow: the effect on technical debt," in *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, 2019, pp. 87–91.

[7] E. Gama, S. Freire, M. Mendonça, R. O. Spinola, M. Paixao, and M. I. Cortes, "Using Stack Overflow to Assess Technical Debt Identification on Software Projects," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Oct. 2020, pp. 730–739. doi: 10.1145/3422392.3422429.

[8] T. Besker, A. Martini, and J. Bosch, "Impact of architectural technical debt on daily software development work—a survey of software practitioners," in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, 2017, pp. 278–287.

[9] T. Besker, A. Martini, and J. Bosch, "Managing architectural technical debt: A unified model and systematic literature review," *Journal of Systems and Software*, vol. 135, pp. 1–16, 2018.

[10] R. Hoda, "Using Agile Games to Invigorate Agile and Lean Software Development Learning in Classrooms."

[11] P. Rodríguez, M. Mäntylä, M. Oivo, L. Lwakatare, P. Seppänen, and P. Kuvaja, "Advances in Using Agile and Lean Processes for Software Development," 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245818300299

[12] W. Cunningham, "A '92 Addendum to the Proceedings Experience Report-The WyCash Portfolio Management System," 1992.

[13] S. Mcconnell, "Managing Technical Debt," 2008. [Online]. Available: www.construx.com/whitepapers

[14] F. Zampetti, C. Vassallo, S. Panichella, G. Canfora, H. Gall, and M. Di Penta, "An empirical characterization of bad practices in continuous integration," *Empir Softw Eng*, vol. 25, no. 2, pp. 1095–1135, Mar. 2020, doi: 10.1007/s10664-019-09785-8.

[15] N. Ozkan and A. K. Tarhan, "Investigating Causes of Scalability Challenges in Agile Software Development from a Design Perspective," in *1st International Informatics and Software Engineering Conference: Innovative Technologies for Digital Transformation, IISEC 2019 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., Nov. 2019. doi: 10.1109/UBMYK48245.2019.8965633.

[16] J. Holvitie *et al.*, "Technical debt and agile software development practices and processes: An industry practitioner survey," *Inf Softw Technol*, vol. 96, pp. 141–160, Apr. 2018, doi: 10.1016/j.infsof.2017.11.015.

[17] T. Dingsoeyr, D. Falessi, and K. Power, "Agile Development at Scale: The Next Frontier," *IEEE Software*, vol. 36, no. 2. IEEE Computer Society, pp. 30–38, Mar. 01, 2019. doi: 10.1109/MS.2018.2884884.

[18] L. Gaikwad and V. Sunnapwar, "The Role of Lean Manufacturing Practices in Greener Production: A Way to Reach Sustainability," *International Journal of Industrial and Manufacturing Systems Engineering*, vol. 5, no. 1, p. 1, 2020, doi: 10.11648/j.ijimse.20200501.11.

[19] S. Sadeghi, A. Akbarpour, and H. Abbasianjahromi, "Provide a Lean and Agile Strategy for an Antifragile Sustainable Supply Chain in the Construction Industry(residential complex)," *Cleaner Logistics and Supply Chain*, vol. 5, Dec. 2022, doi: 10.1016/j.clscn.2022.100079.

[20] N. Brown *et al.*, "Managing technical debt in software-reliant systems," in *Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research, FoSER 2010*, 2010, pp. 47–51. doi: 10.1145/1882362.1882373.

[21] N. Rios, R. O. Spínola, M. Mendonça, and C. Seaman, "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," *Empir Softw Eng*, vol. 25, no. 5, pp. 3216–3287, Sep. 2020, doi: 10.1007/s10664-020-09832-9.

[22] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering - A systematic literature review," *Information and Software Technology*, vol. 51, no. 1. pp. 7–15, Jan. 2009. doi: 10.1016/j.infsof.2008.09.009.

[23] F. Tian, P. Liang, and M. A. Babar, "How developers discuss architecture smells? An exploratory study on stack overflow," in *Proceedings - 2019 IEEE International Conference on Software Architecture, ICSA 2019*, Institute of Electrical and Electronics Engineers Inc., Apr. 2019, pp. 91–100. doi: 10.1109/ICSA.2019.00018.

[24] F. Tian, P. Liang, and M. A. Babar, "How developers discuss architecture smells? An exploratory study on stack overflow," in *Proceedings - 2019 IEEE International Conference on Software Architecture, ICSA 2019*, Institute of Electrical and Electronics Engineers Inc., Apr. 2019, pp. 91–100. doi: 10.1109/ICSA.2019.00018.

[25] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *Journal of Systems and Software*, vol. 150, pp. 77–97, Apr. 2019, doi: 10.1016/j.jss.2019.01.001.
[26] T. Sharma, "How Deep is the Mud: Fathoming Architecture Technical Debt Using Designite", doi: 10.5281/zenodo.2566832.
[27] R. L. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas, "In search of a metric for managing architectural technical debt," in *Proceedings of the 2012 Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture, WICSA/ECSA 2012*, 2012, pp. 91–100. doi: 10.1109/WICSA-ECSA.212.17.
[28] A. Walker, D. Das, and T. Cerny, "Automated code-smell detection in microservices through static analysis: A case study," *Applied Sciences (Switzerland)*, vol. 10, no. 21, pp. 1–20, Nov. 2020, doi: 10.3390/app10217800.
[29] L. Sousa, W. Oizumi, A. Garcia, A. Oliveira, D. Cedrim, and C. Lucena, "When are smells indicators of architectural refactoring opportunities? a study of 50 software projects," in *IEEE International Conference on Program Comprehension*, IEEE Computer Society, Jul. 2020, pp. 354–365. doi: 10.1145/3387904.3389276.
[30] T. Bi, P. Liang, and A. Tang, "Architecture Patterns, Quality Attributes, and Design Contexts: How Developers Design with Them?"
[31] S. Martinez-Fernandez *et al.*, "Continuously Assessing and Improving Software Quality with Software Analytics Tools: A Case Study," *IEEE Access*, vol. 7, pp. 68219–68239, 2019, doi: 10.1109/ACCESS.2019.2917403.
[32] P. Avgeriou *et al.*, "An Overview and Comparison of Technical Debt Measurement Tools."
[33] O. Lehti, "State, reasons, and effects of technical debt in a Finnish IoT-domain software project."
[34] L. Rosser, "A Systems Perspective on Technical Debt," 2021.
[35] R. Verdecchia, P. Kruchten, P. Lago, and I. Malavolta, "Building and evaluating a theory of architectural technical debt in software-intensive systems," *Journal of Systems and Software*, vol. 176, Jun. 2021, doi: 10.1016/j.jss.2021.110925.
[36] A. Rasheed *et al.*, "Requirement Engineering Challenges in Agile Software Development," *Mathematical Problems in Engineering*, vol. 2021. Hindawi Limited, 2021. doi: 10.1155/2021/6696695.
[37] C. Research, N. Rios, M. Mendonça, C. Seaman, and R. Oliveira Spínola, "Technical Debt Causes and Effects in Agile Projects Causes and Effects of the Presence of Technical Debt in Agile Software Projects," 2019.
[38] S. Freire *et al.*, "Pitfalls and Solutions for Technical Debt Management in Agile Software Projects," *IEEE Softw*, vol. 38, no. 6, pp. 42–49, 2021, doi: 10.1109/MS.2021.3101990.
[39] A. S. E. Hamed, H. M. Elbakry, A. E. Riad, and R. Moawad, "A Proposed Technical Debt Management Approach Applied on Software Projects in Egypt," *Journal of Internet Services and Information Security*, vol. 13, no. 3, pp. 156–177, Aug. 2023, doi: 10.58346/JISIS.2023.I3.010.
[40] A. Almeida Da Costa Júnior, "A Maturity Model based on ISO/IEC/IEEE 42010:2011 to Identify Technical Debt in Software Architecture."

[41] B. S. Aragão *et al.*, "TestDCat: Catalog of Test Debt Subtypes and Management Activities," pp. 279–295, 2019, doi: 10.1007/978-3-030.

[42] M. Wiese, P. Rachow, M. Riebisch, and J. Schwarze, "Preventing technical debt with the TAP framework for Technical Debt Aware Management," *Inf Softw Technol*, vol. 148, Aug. 2022, doi: 10.1016/j.infsof.2022.106926.

[43] L. Xavier, R. Dos Santos, and M. T. Valente, "Agile Technical Debt Management using the LTD Framework," 2022.

[44] M. G. Stochel, T. Borek, M. R. Wawrowski, and P. Chołda, "Business-driven technical debt management using Continuous Debt Valuation Approach (CoDVA)," *Inf Softw Technol*, vol. 164, Dec. 2023, doi: 10.1016/j.infsof.2023.107333.