Middle East Technical University
Informatics Institute

IDENTIFYING TECHNICAL DEBT AND TOOLS FOR TECHNICAL DEBT MANAGEMENT IN SOFTWARE DEVELOPMENT

Advisor Name: Prof. Dr. Altan KOÇYİĞİT
(METU)

Student Name: Tolga MURATDAĞI
(Software Management - SM 589)

January 2024

TECHNICAL REPORT
METU/II-TR-2024-

Orta Doğu Teknik Üniversitesi
Enformatik Enstitüsü

YAZILIM GELİŞTİRMEDE TEKNİK BORÇ TANIMLAMA VE TEKNİK BORÇ YÖNETİMİ ARAÇLARI

Danışman Adı: Prof. Dr. Altan KOÇYİĞİT
(ODTÜ)

Öğrenci Adı: Tolga MURATDAĞI
(Yazılım Yönetimi – SM 589)

Ocak 2024

TEKNİK RAPOR
ODTÜ/II-TR-2024-

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY (Internal Use)	**2. REPORT DATE** 19.01.2024
3. TITLE AND SUBTITLE **IDENTIFYING TECHNICAL DEBT AND TOOLS FOR TECHNICAL DEBT MANAGEMENT IN SOFTWARE DEVELOPMENT**	
4. AUTHOR (S) Tolga MURATDAĞI	**5. REPORT NUMBER (Internal Use)** **METU/II-TR-2024-**
6. SPONSORING/ MONITORING AGENCY NAME(S) AND SIGNATURE(S) Software Management Master's Programme, Department of Information Systems, Informatics Institute, METU Advisor: Prof. Dr. Altan KOÇYİĞİT	Signature:
7. SUPPLEMENTARY NOTES	

8. ABSTRACT (MAXIMUM 200 WORDS)

This term project explores the concept of technical debt in software development, as initially articulated by Ward Cunningham in 1992. Technical debt is a multifaceted compromise that involves finding a balance between speed and the necessity for future changes. The study classifies many types of debt that occur at different stages of the software development life cycle, including complexity at the code level, challenges in design, and compromises in architecture. At the same time, it assesses specialist tools such as visualization, dynamic analysis, and static analysis tools that are designed to facilitate efficient debt management. This research takes a different approach compared to previous studies by providing a full review of technical debt management methods that are commonly used and can be applied at every stage of software development. The study provides comprehensive information on fundamental concepts, methods for recognizing technical debt, and evaluations of tools. It is a significant asset for organizations dealing with the complexities of technical debt, enabling them to make well-informed decisions in software development.

9. SUBJECT TERMS Technical Debt, Technical Debt Management Tools, Software Development Life Cycle	**10. NUMBER OF PAGES** 71

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This project focuses on the urgent necessity to comprehend and alleviate technical debt in software development. Technical debt, coined by Ward Cunningham in 1992, refers to the complex balance between the speed of software development and the need for future changes.

The project aims to achieve two main objectives. Firstly, it involves classifying different types of technical debt, including code-level complexities, design obstacles, and architectural compromises, that occur during the software development life cycle. Secondly, it involves assessing specialized tools, such as visualization, dynamic analysis, and static analysis tools, that are specifically designed to effectively manage technical debt.

This research adopts an integrated approach, offering a thorough review of widely-used technical debt management solutions that may be applied at every step of software development. Unlike earlier studies that generally concentrate on specific life cycle stages or individual tools, this research takes a broader perspective.

The study provides a comprehensive overview of technical debt, including its fundamental concepts, an examination of different debt types and methods for identifying them, and an analysis of management strategies, including their criteria, benefits, and drawbacks. The literature review situates the research within the wider academic environment, highlighting the comprehensive viewpoint.

The project culminates with an evaluation of tool selection, describing the work that has been accomplished, and highlighting the main points of agreement. This research is a significant resource for firms who want to make informed decisions in order to manage and reduce technical debt over the software development life cycle.

Keywords: Technical Debt, Technical Debt Management Tools, Software Development Life Cycle

CHAPTER 1

INTRODUCTION

Many businesses find the objective of comprehending and controlling technological debt to be appealing. Taking proactive measures to manage technical debt offers organizations the opportunity to effectively manage the expenses associated with making changes, by seamlessly integrating technical decision-making and software economics with software engineering delivery [4].

The concept of Technical debt (TD) was initially introduced by Ward Cunningham in 1992 as a metaphor to illustrate the intricate trade-off between speed and the need for future revision when striving to produce high-quality software. In other words, it is a broad word that encompasses many flaws and deficiencies in software, resulting in the need for more maintenance work [17].

In the realm of software development, the existence of technical debt is unavoidable and might even be seen as advantageous in order to attain immediate advantages. Managing the expense of TD can lead to profitability. Hence, it is crucial to maintain strict control over the accrued debts [4]. The goal of technical debt management (TDM) in this context is to facilitate informed decision-making regarding the necessity of addressing a technical debt item and determining the optimal timing for doing so. Over the past few years, it has arisen as a new research field [7]. It comprises a sequence of actions aimed at avoiding the accumulation of undesirable technical debt or managing existing debt to ensure it remains below acceptable limits. Efficiently managing TD necessitates the utilization of tools. Academia and industry have recently put forth many strategies for effectively managing technical debt in software projects [8].

This term project is driven by the need to understand and reduce the negative effects of technical debt. It specifically focuses on two connected areas: identifying technical debt and exploring the array of tools available for the effective management of it.

The primary objective of the project is, to investigate and categorize the various forms of technical debt that arise throughout the software development life cycle. These debts consist of complexities at the code level, obstacles in design, and compromises in architecture. And the second objective is, to examine and assess a variety of instruments that are specifically engineered to detect and control technical debt. This consists of visualization tools, dynamic analysis tools, and static analysis tools, each of which offers a distinct viewpoint on the reduction of technical debt.

When we look at the academic studies carried out, it can be seen that many studies have been studied on the subject of "technical debt and technical debt management tools" in recent years. However, these studies are mainly specialized in a specific area in the software life cycle, and it is seen that studies conducted specifically on tools are generally conducted on a single tool. The main purpose of this study and its difference from these studies is, defining the technical debt issue from an integrated perspective, compile the most used technical debt management tools that can be used at every stage in the software life cycle, and present the results by evaluating them within the scope of various criteria.

The report begins with an overview of technical debt (TD), followed by an examination of specific categories of technical debt, the repercussions and effects of TD, and techniques and approaches for identifying technical debt. Then, comprehensive information regarding the instruments utilized for technical debt management, including their primary criteria, advantages, and disadvantages. Following this, the relevant literature is discussed, detailing the research. Following this are the considerations for tool selection in technical debt management, and the report concludes with a summary of the completed work and the consensus reached.

CHAPTER 2

BACKGROUND

In this section, I present the definition of the technical debt term and its relation with the software project lifecycle. Also, types of technical debts, consequences and impacts of it, methods and strategies to identify the technical debt especially the tools that are used in technical debt management.

2.1. What Is Technical Debt

By employing a financial analogy, the notion of technical debt reframes the discussion on decision making, moving it away from purely technical or economic considerations [3]. This allows developers and managers to gain a clearer understanding of the trade-offs and concessions involved in software development, enabling them to make informed decisions about the future course of action [8]. This section provides an overview of the technical debt landscape by examining the many manifestations of technical debt in different types of development artifacts throughout the software development lifecycle.

2.1.1. Technical Debt Landscape

Figure 1 depicts a standard technical debt landscape, showcasing the software development challenges that engineers address in order to enhance the system [25]. We differentiate between the apparent concerns, such as demands for additional features and defects requiring resolution, and the predominantly imperceptible ones, which are only discernible to software developers. The figure predominantly displays concerns pertaining to evolution on the left side, while issues concerning upkeep and quality are predominantly shown on the right side [25].

The emphasis is placed on the largely invisible elements of evolution and maintenance. Diverse categories of development artifacts, including the code, the architecture, and the production infrastructure, accumulate technical debt in distinct ways [25].

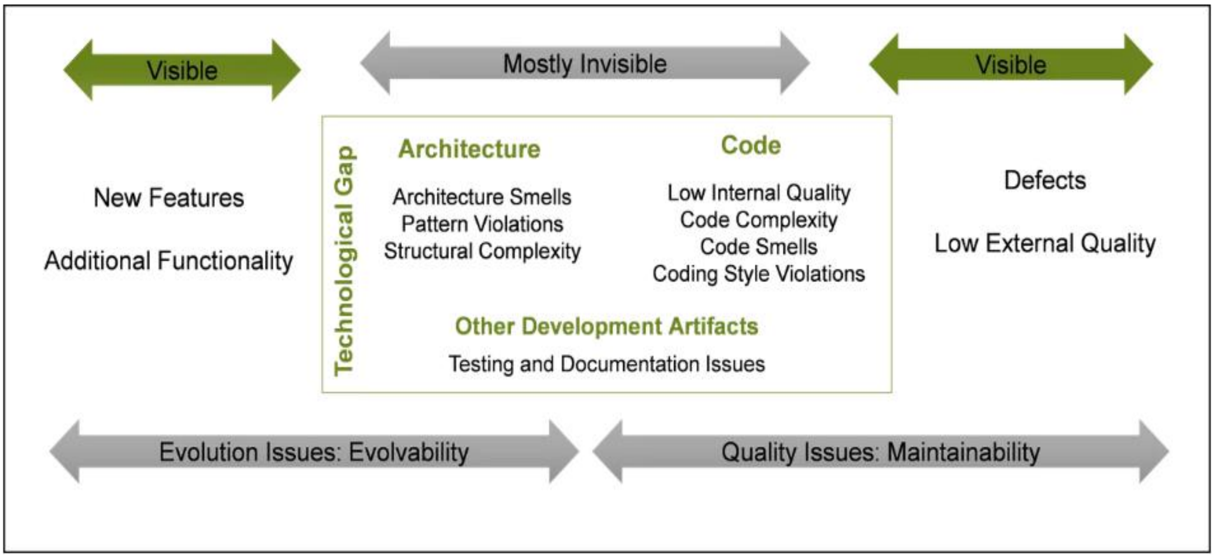

Figure 1: *The technical debt landscape [25]*

Static checkers can be utilized to subject the code to examination, scrutiny, and evaluation in order to detect issues of a more minute scale, such as coding standard violations, incorrect or misleading comments, code clones, and superfluous code complexity. A number of these technical debt symptoms are commonly known as "code smells." When a system accumulates technical debt at the source code level, it typically impedes maintainability, thereby complicating the process of implementing necessary system corrections [9].

Additional technical debt items are more extensive and ubiquitous. These decisions pertain to the configuration or architecture of the system. Some well-known technical debt symptoms related to architecture are architecture smells, pattern violations and structural complexity [8].

Finally, certain technical debt items are linked to the code of closely related software production artifacts, such as test suites, build scripts, or deployment infrastructure, rather than the product's code [9].

2.1.2. Technical Debt Timeline

Over time and as the software system evolves, technical debt becomes increasingly significant [8]. In the hypothetical scenario where the system remains static, the obligation to repay interest would be null and void, rendering technical debt inconsequential. Figure 2 illustrates how technical debt develops over time.

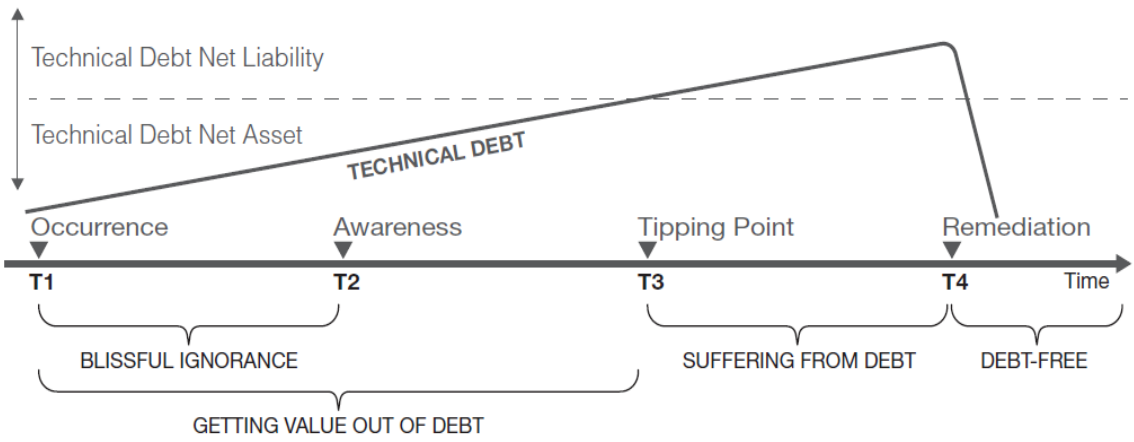

Figure 2: *The technical debt timeline [25]*

The moment a technical debt item is introduced into the system, for whatever positive or negative purposes, is referred to as its “Occurrence” (T1). At “Awareness” (T2) point, the development organization begins to observe the technical debt item's symptoms. The interval between T1 and T2 is characterized by a state of blissful unawareness. At “Tipping Point” (T3), the expenses associated with technical debt begin to surpass the initial benefits derived from incurring said debt. Prior to T3, one might as well as live with the technical debt because it provides some benefit. However, following the T3, you should now pay more than you gain. Finally at the “Remediation” point, the remediation cost comprises the principal amount as well as all interest that has been accrued. Therefore, remediation frequently requires more effort than simply reversing the incorrect code and implementing the correct solution at T1. The remediation may result in a substantially different design than the one you abandoned at T1 due to the substantial evolution of the context [25].

2.1.3. Relationship between Technical Debt and Source Code

Comprehending the connection between technical debt and the source code is crucial for understanding the influence of development approaches on software quality and maintainability. Technical debt commonly appears in the source code of a software application, serving as a prominent and observable sign of the accumulated debt [1].

Here are key points highlighting the relationship between technical debt and the source code:

- Manifestation in Code Quality [16]:

Technical debt is frequently accumulated during the development process when developers make compromises or employ expedient solutions to meet imminent deadlines. These concessions can lead to code quality that is less than optimal.

Code-level technical debt include problems such as code smells, duplications, intricate code architectures, and other deviations from optimal coding standards.

- Readability and Maintainability [1]:

Technical debt present in the source code can have a negative impact on the codebase's readability and maintainability. Inadequately composed or intricate code poses difficulties for developers in comprehending and altering the code in subsequent instances.

Accumulated technical debt can result in a codebase that is challenging to navigate, impeding the effectiveness of development and maintenance tasks.

- Impact on Bug Rates [16]:

Technical debt in the source code is frequently linked to a higher probability of producing software defects. Unresolved code-level problems can lead to a greater occurrence of flaws and complications throughout the software development process and in the final output.

- Refactoring and Technical Debt Reduction [13]:

Refactoring is a prevalent approach used to tackle technical debt at the code level. Code refactoring is the process of reorganizing the source code to enhance its quality, readability, and maintainability, while keeping its exterior behavior unchanged.

Efficient refactoring aids in diminishing technical debt, enhancing the resilience of the source code, and aligning it with coding standards.

- Continuous Monitoring and Improvement [15]:

Regularly monitoring the source code is essential for detecting and preventing the building of rising technical debt. Code quality metric analysis tools facilitate continuous improvement initiatives.

Through proactive management of technical debt in the source code, development teams may guarantee the codebase's long-term adaptability and maintainability.

- Documentation and Comments [15]:

The technical debt present in the source code encompasses more than simply the functional features. Additionally, it encompasses the documentation and comments included within the codebase.

Insufficient or obsolete documentation inside the source code leads to gaps in understanding, highlighting the importance of addressing technical debt connected to documentation.

To summarize, technical debt and the source code are closely interconnected. The choices taken throughout the coding process, the compromises between efficiency and excellence, and the methodologies adhered to by developers are all evident in the source code. Tackling technical debt at the source code level is essential for ensuring a robust and enduring software development process. Regular code reviews, refactoring, and a dedication to coding best practices are essential components in efficiently controlling technical debt inside the source code.

2.1.4. Relationship between Technical Debt and Architecture

Architectural technical debt refers to the metaphorical burden caused by major design decisions, such as those related to structure, frameworks, technologies, and languages, that may have been appropriate or even ideal at the time they were made, but ultimately impede future advancement. Unlike code-level technical debt, which can be easily recognized and refactored with minimal work, architectural debt is challenging to detect, has a wide range of costly remedies, is intimidating, and is typically deliberately avoided [1].

Architectural technical debt elements have a significant influence on the entire system as they are intricately connected in a complicated web of interdependencies. Poorly conceived architecture leads to cost accumulation as the system becomes increasingly difficult to adapt. Modifying fundamental architectural decisions can prove significantly more challenging than modifying source code, particularly as the system expands, due to the extensive repercussions such changes entail. Remediation is a significant endeavor that may extend across numerous iterations or deplete a substantial portion of the available resources throughout multiple releases [16].

A well-designed architecture that is followed during system implementation directly correlates with a controllable buildup of technical debt. For example, if the objective is to maintain the system for several decades and adapt to evolving technology, the system's architecture should facilitate the division of responsibilities, employ independent technology layers for effortless upgrades, and guarantee that modifications are confined to facilitate the addition of new features. These architecture considerations are crucial and should guide the design reviews and be evident in the coding, not just at the start of the system's development but throughout its entire lifecycle [18].

The system must to be meticulously crafted and supervised to ensure compliance with "quality attributes," which refer to architecturally critical requirements pertaining to the system's reliability, security, and maintainability. Quality characteristics direct attention towards the interrelated parts of the system, including its performance under varying circumstances, data flow and management, and its reliance on other software components such as databases, user interface and backend frameworks, middleware, etc [16].

There are multiple approaches you can employ to identify technical debt in the system's design while going through the various stages of technical debt analysis. Generally, the most effective strategy involves a blend of these activities [25]:

- Inquire with the designers regarding the system's well-being or any issues it may be experiencing.
- Analyze the structure of the architecture.
- Analyze the code to have a deeper understanding of the underlying structure.

2.1.5. Relationship between Technical Debt and Production

The production phase of the software development process involves the following four activities; build (create the executable software), system tests (validate that software is ready), deployment and open it.

So, we can handle the technical debt in production phase in three main categories:

- Build and Integration Debt:

Inadequate or improper design and coding of the build scripts themselves: Build scripts are essentially lines of code, occasionally aided by specific code included into the application being developed [8].

Misalignment between the build dependencies and the actual code: Due to the rapid evolution of the program, newly introduced components may lack backward compatibility [9].

- Testing Debt:

Inadequate or improper design and coding of tests: Test suites are essentially code and are occasionally aided by specific code integrated into the developing program. Extensive collections of automated tests may lack a distinct objective; when they encounter failure, it is likely that something is amiss, but it is uncertain which elements caused the failure and the underlying reasons behind it [9].

Misalignment between the tests and the actual code: Due to the rapid evolution of software, there is a possibility that new tests may be absent or may only assess an outdated understanding of the requirements. Tests that are highly detailed and implemented early in the development process, particularly when using mockup software, can become difficult to maintain due to the intricate connections they build with the production code. A little modification could result in the failure of lots of tests [8].

Challenges of SaaS (Software as a Service) contexts: The alignment of development, testing, and production environments can get disrupted. If developers utilize version X, the continuous integration system should be version Y, and the production servers should be version Z. If these conditions are not met, the tests being conducted may not be targeting the correct elements, and the developers may be unaware of this discrepancy. Alternatively, a code that functioned flawlessly throughout the development phase may encounter issues when implemented in the testing infrastructure [25].

- Infrastructure Debt:

In the structure of the operational system: Within the framework of the operational system, one potential issue is the absence of "observability," also known as monitoring debt [8].

In scripts: This may involve scripts that implement the deployment of the code, the data, and the updates on the operational system [9].

The absence of verification for deployment scripts contributes to the accumulation of technical debt. Verifying the compatibility of scripts with the architecture is crucial in order to prevent inconsistencies between development, testing, and production environments and to reduce potential risks. [25]

2.2. Types of Technical Debt

Technical debt may present itself in a multitude of manifestations during the course of software development. Different categories of technical debt can be distinguished according to their characteristics and origins. Technical debt can be categorized into two main types [15]:

- "Unintentional TD," which is characterized by involuntary and nonstrategic movements, is frequently attributed to inadequately planned operations resulting from the presence of inexperienced personnel or alterations in the environment.
- "Intentional TD" refers to the purposeful and planned decision-making by professionals to seek short-term benefits by taking shortcuts, considering other options, and leaving projects unfinished.

In addition, TD can manifest in various activities and stages of the software development life cycle. With this, according to research in literature 10 different and most seen types of identified technical debts is shown at Table 1 [2].

Type of TD	Definition
Code Level	Code-level technical debt encompasses any poor or compromised elements included in the source code of a software application. This encompasses suboptimal or inadequately organized code, code with undesirable characteristics, redundant code, and other problems that, although they may offer a temporary resolution, can impede the long-term maintainability, readability, and general excellence of the codebase [2].
Design Level	Design-level technical debt refers to compromises or deficiencies in the overall architecture and design elements of a software product. It encompasses suboptimal design choices, architectural weaknesses, and patterns that may have been convenient in the short run but can provide difficulties in terms of scalability, adaptability, and long-term system maintainability [4].
Architecture Level	Architecture-level technical debt refers to use of outdated technologies, frameworks, or platforms, leading to potential security vulnerabilities and maintenance challenges. Also, architectural choices that hinder the system's ability to scale, resulting in performance bottlenecks and limitations under increased loads [1].
Requirements Level	Requirements-level technical debt refers to compromises or deficiencies in the definition and documentation of software requirements. It arises when the requirements are ambiguous, inadequate, or prone to frequent alterations, resulting in difficulties in the process of creation, testing, and maintenance. It is crucial to address technical debt at the requirements level to ensure that the software matches successfully with stakeholder expectations and project goals [15].

Testing and Quality Assurance	Technical debt at the Testing and Quality Assurance level pertains to compromises or shortcomings in the processes of testing and quality assurance inside a software project. These factors may encompass insufficient test coverage, postponed testing operations, and the existence of unattended flaws or vulnerabilities. Technical debt at the testing and quality assurance (QA) level can have negative effects on program reliability, raise the likelihood of faults, and impede the overall quality of the software product [2].
Documentation	Documentation-level technical debt refers to compromises or flaws in the documentation of a software project. This may encompass documentation that is insufficient, outdated, or poorly organized, hence posing difficulties for developers and stakeholders in comprehending, maintaining, and expanding the product. Resolving technical debt at the documentation level is essential for promoting clear communication, facilitating knowledge exchange, and assuring the long-term sustainability of the software [10].
Deployment and Infrastructure	Deployment and Infrastructure level technical debt pertains to compromises or shortcomings in the procedures and infrastructure associated with the deployment and upkeep of a software program. This could entail the utilization of obsolete deployment methodologies, non-scalable infrastructure, or ineffective configuration management. To ensure seamless and effective deployment procedures, scalability, and general stability of software in production environments, it is crucial to tackle technical debt at the deployment and infrastructure levels [9].
Security Level	Security-level technical debt refers to compromises or deficiencies specifically pertaining to the security aspects of a software product. These risks encompass unpatched

	vulnerabilities, insufficient encryption techniques, and other security-related concerns. If left unattended, they can jeopardize the confidentiality, integrity, and availability of the software. It is essential to prioritize the resolution of security-related technical debt in order to protect the software against potential risks and weaknesses [8].
Knowledge and Skill (People)	People technical debt, in the context of software development, pertains to the shortcomings or weaknesses in the knowledge and abilities possessed by the individuals participating in the process. These factors may encompass inadequate training, limited expertise in specific technologies, or a knowledge deficit within the development team. To tackle knowledge and skill technical debt, it is required to allocate resources towards training programs, mentorship initiatives, or recruitment of persons possessing the requisite skills, in order to improve the overall capabilities [15].
Process Level	Process-level technical debt refers to compromises or deficiencies in the established processes and methodologies used in software development. This may involve shortcuts or suboptimal practices in project management, development workflows, or quality assurance processes. Addressing process-level technical debt is crucial for optimizing efficiency, improving collaboration, and ensuring that the development team follows best practices throughout the software development lifecycle [13].

Table 1: *Types of Technical Debt*

2.3. Consequences and Impacts of Technical Debt

Technical debt can result in various outcomes and influences on software development initiatives and the general well-being of a software system. The following are crucial factors that emphasize the repercussions and influence of technical debt:

- *Increased Development Time:* Resolving technical debt frequently necessitates allocating extra time. As the debt increases, developers may allocate additional time towards resolving defects, restructuring code, or finding workarounds for inadequately designed components. This can impede the overall progress of development [12].
- *Reduced Developer Productivity:* Developers faced with a codebase encumbered with technical debt may encounter heightened difficulties in writing new code, comprehending old code, or executing modifications with efficiency. The decrease in productivity might result in team members experiencing frustration and burnout [13].
- *Higher Maintenance Costs:* The expenses associated with sustaining a system burdened by technical debt are often higher. The complexity and time required for bug repairs, upgrades, and modifications escalate, resulting in a greater maintenance burden on development teams [15].
- Quality Compromises: Technical debt frequently leads to trade-offs in code quality. Compromising on quality to meet strict time constraints or solve immediate requirements might result in inferior solutions, which in turn can make the codebase more challenging to maintain, comprehend, and expand [14].
- *Increased Bug Count:* Technical debt is strongly correlated with a higher incidence of defects in a software system. Code that is poorly conceived or developed hurriedly is more susceptible to errors and faults, resulting in a greater number of bugs that must be handled in the long run [13].
- *Risk of Project Failure:* Insufficient management of technical debt can lead to its accumulation, reaching a critical level that jeopardizes the project's success. The system has the potential to become too intricate, unstable, or challenging to sustain, hence endangering the project's overarching goals [14].
- *Impact on User Experience:* The presence of technical debt might have a detrimental effect on the overall user experience. Technical debt can cause performance issues,

unanticipated failures, and system downtimes, which in turn can negatively impact the user experience, leading to decreased customer satisfaction and retention [12].

- *Security Risks:* Technical debt may give rise to security risks, including obsolete libraries, unresolved issues, or insecure coding methodologies. These vulnerabilities constitute a significant threat to the system's security, potentially leading to data breaches, illegal access, and other security problems [13].
- *Long-Term Maintenance Issues:* Prolonged maintenance issues might arise from the accumulation of technical debt over time if left unattended. Legacy systems that have accumulated a significant amount of technical debt can become challenging and costly to operate, potentially necessitating a substantial overhaul or redesign [14].
- *Negative Impact on Innovation:* Technical debt might hinder the progress of innovation within a development team. The allocation of resources towards correcting technical debt may impede the ability to innovate and maintain competitiveness, diverting them from potential new features or enhancements [13].
- *Reduced Team Morale:* The presence of extensive technical debt in a codebase can have a negative effect on the overall morale of the team. Developers may experience frustration due to the persistent requirement to resolve issues and may lose motivation if they sense that the codebase is not progressing [12].

To summarize, technical debt has a wide range of ramifications and effects on several areas of software development, including the effectiveness of development processes and the ultimate success and longevity of a software system. Effectively managing and reducing technical debt is crucial for ensuring the sustainability and efficiency of a development ecosystem.

2.4. Methods and Strategies for Identifying Technical Debt

Within the continually evolving domain of software development, the identification and management of technical debt play a crucial role in guaranteeing the long-term viability, maintainability, and efficiency of a codebase. This involves utilizing different approaches and procedures to uncover and resolve problematic regions [10].

Detailed information about different methods and strategies for identifying technical debt which are the most used in literature and industry.

- Code Reviews [15]:

Code reviews entail a cooperative analysis of the source code by team members to detect any problems pertaining to coding standards, design patterns, and code quality. During code reviews, engineers have the ability to identify specific instances when expedient measures were employed, resulting in possible accumulation of technical debt. Discoveries of inconsistencies, non-compliance with best practices, and complications frequently yield valuable insights about the condition of the codebase.

- Static Code Analysis [11]:

Static code analysis is the process of utilizing automated techniques to examine source code without actually running it. These tools are capable of identifying coding patterns, anti-patterns, and possible problems such as code smells or security vulnerabilities. SonarQube and ESLint are software tools that conduct static analysis, enabling teams to detect technical debt by highlighting instances of coding standards violations and probable problematic regions.

- Dynamic Analysis [15]:

Dynamic analysis entails evaluating the operational behavior of an application during its execution. Profiling tools, such as VisualVM, can be utilized to detect performance bottlenecks, memory leaks, and other runtime issues that contribute to technical debt. This approach is especially efficient in identifying problems that may not be evident during static analysis or code reviews.

- Architectural Analysis [10]:

Performing regular reviews of the architecture and design of a software system can help uncover instances when architectural decisions may have contributed to the accumulation of technical debt. Architecture review sessions or the utilization of tools such as Structure101 and Sonargraph aid in assessing the general condition of the architecture and revealing possible technical debt.

Each of these methodologies and tactics adds to a holistic approach for identifying and resolving technical debt inside a software development project. By combining these strategies, teams can obtain a comprehensive understanding of the codebase and make well-informed judgments regarding the effective prioritization and mitigation of technical debt.

- Automated Testing [13]:

Insufficient or deficient test coverage can suggest the presence of technical debt. Automated testing technologies, such as JUnit for unit testing and Selenium for UI testing, assist in identifying sections of the codebase that are inadequately tested. Inadequate test coverage can indicate heightened risk and the possibility of accruing technical debt when modifying the code.

- Documentation Review [15]:

Examining documentation, or the absence of it, is a technique for identifying technical debt associated with knowledge transfer and maintainability. Obsolete or absent documentation might result in misinterpretations and challenges in managing the codebase. Doxygen or Sphinx can be utilized to automatically produce and manage documentation.

- Monitoring and Logging [11]:

Consistently monitoring application logs and performance metrics might reveal any flaws that may affect the dependability and efficiency of a system in operation. Log analysis technologies such as the ELK Stack (Elasticsearch, Logstash, Kibana) or Prometheus assist in the identification and comprehension of runtime issues and their impact on technological debt.

- User Feedback and Bug Reports [15]:

Engaging in the proactive collection and analysis of user feedback, in addition to bug reports, is an invaluable method for discovering and addressing system issues. During the development process, users frequently come across issues that are not immediately obvious. Their input can reveal elements of technical debt that impact the user experience.

To summarize, employing tools to handle technical debt is essential for promoting a proactive and methodical approach to upholding software quality and sustainability. Industry statistics demonstrate an increasing dependence on these tools, as surveys indicate that more than 80% of software development teams integrate automated analysis tools into their workflows.

CHAPTER 3

RELATED WORK

Many types of papers and blogs are authored by individuals from academia and experts entrenched in the industry, separately describing what the technical debt is, the importance and consequences of technical debt in software projects and some tools that are used for managing technical debt. Nevertheless, a comprehensive analysis is required that integrates all the information from the publications and blogs. The work may have become obsolete due to the rapid development and improvement of tools in the industry.

Through an extensive review of the existing literature, it becomes evident that there are numerous factors to take into account and evaluate when choosing a method for managing technical debt. No instrument can be designated as "mandatory" since none possesses the highest level of dominance in every aspect. In the following, the recent related work are introduced and they are also summarized in Table 4.

In 2018, a study was done by a paper that specifically examines the literature pertaining to architectural technical debt [1]. The authors chose and examined 47 source publications in order to analyze and describe the strategies used for identifying architectural technical debt (ATD). This analysis focused on publishing patterns, the characteristics of these techniques, and their potential for being adopted in industrial settings. Their research reveals potential avenues for future investigation in the field of ATD, including the utilization of the temporal aspect in ATD identification and the subsequent resolution of ATD. The authors emphasize the necessity for more industrial participation in the formulation, design, and evaluation of ATD identification approaches.

In 2019, Lenarduzzi et al. published a systematic literature review on the prioritization of technical debt [2]. The study analyzed 37 carefully chosen studies that encompass the most advanced methods, criteria, metrics, and tools employed in both practical and research settings to prioritize technical debt. They have discovered seven strategies that specifically target the prioritizing of technical debt. The primary finding of their study is the absence of agreement regarding the crucial elements to prioritize TD and the appropriate methods to assess them. Their findings indicate that code and architectural debt are the most extensively studied forms of debt when prioritizing. The investigation also verified the absence of a robust, validated, and generally adopted toolkit specifically designed for prioritization.

Another research in the area was done in 2017, conducted as a systematic literature review to examine the concept of technical debt in the context of agile software development [3]. The authors made a comprehensive analysis of 38 papers. Their objective was to identify specific study areas of interest, classify the causes and effects of TD, and determine effective management strategies and tools. The “DebtFlag” and “NDepend” were cited as tool for identifying technical debt in source code during agile development. Their findings suggest a requirement for more tools, models, and standards to facilitate the management of technical debt in agile development.

In 2015, a study was done to systematically map and provide an overview of the existing research on technical debt management [4]. This study encompassed various activities, methodologies, and instruments associated with the topic. They identified a compilation of 10 forms of technical debt, 8 actions for managing technical debt, and 29 instruments for managing technical debt that were derived from research investigations. Technical debt tools provide information on their functionality, vendor, categories of technical debt, and the artifacts they handle. The research suggests that there is a need for additional specialized TD management solutions. They determined that only 4 out of the total of 29 instruments are specifically designed for TD management. The remaining 25 tools are modified for TD identification, drawing from other fields of software development such as static analysis tools or code smell detection techniques.

In 2020, Avgeriou et al. published a review of the current state of TD tools, with a specific focus on tools that assist in quantifying technical debt [5]. Their research is focused solely on analyzing code, design, and architectural technical debt. Their research concentrated on a collection of 9 software analysis tools: “CAST”, “Sonargraph”, “NDepend”, “SonarQube”, “DV8”, “Squore”, “CodeMRI”, “Code Inspector”, and “SymfonyInsight”.

In 2021, Saraiva et al. made a systematic mapping study, explored the technical debt tools by determining the specific activities, features, and types of technical debt that are addressed by current tools designed to assist in managing technical debt in software projects [6]. Their research found a total of 50 instruments for Technology Development. The majority of these technologies focus on resolving technical debt associated with code, design, and/or architecture artifacts. Based on their research, tools that handle the detection and measurement of TD are still the most common. Nevertheless, it has been noted that current approaches that concentrate on the prevention, replacement, and prioritizing of TD activities are indicative of emerging research patterns.

Another study made in 2020 is about assessing the coherence of the utilization of technical debt language and its alignment with the established conceptual framework [7]. Furthermore, the paper explores the extent to which the metaphorical origins of the phrase "technical debt" persist and impact the research. The study's findings indicate that there is still ambiguity surrounding the origin of metaphorical expression of technical debt in research, and it is necessary to reduce this ambiguity. Tool designers, like “SonarQube”, are not constrained by research findings and can contribute to further ambiguity in defining technological debt. Furthermore, decision makers should utilize risk management models to facilitate the management of technological debt. Hence, the Architecture Tradeoff Analysis Method and other Quality Attribute Models can be utilized as valuable resources to enhance the existing technical debt model.

In 2018, BenIdris et al. published a systematic mapping study. The goal of this study is classifying TD types and showing the indicators used to detect TD and finding the estimators used to quantify the TD, evaluating how researchers investigate. Authors, presented the most common indicators and evaluators to identify and evaluate the TD, and they gathered thirteen types of TD [8].

In 2021, a study was done to assess a system that prioritizes the avoidance and repayment of TD. The technology was created and implemented within the information technology division of a publishing company. The distinctive aspect of this approach lies in the incorporation of TD management within project management. The evaluation was conducted through a study that utilized ticket statistics and a structured survey including people from both the observed IT unit and a comparison unit. The evaluation demonstrates that implementing this paradigm enhances awareness of the occurrence of Technical Debt [9].

Study	Author(s)	Year	Brief
Architectural technical debt identification: The research landscape	Verdecchia, R. et al.	2018	This research use the systematic mapping study approach to identify, classify, and evaluate the current status of architectural technical debt identification. The study focuses on three perspectives: publishing trends, characteristics, and potential for industry adoption.
Technical Debt Prioritization: State of the Art. A Systematic Literature Review	Lenarduzzi, V. et al.	2019	The objective of this study is to examine the current knowledge in software engineering in order to comprehend the many Technical Debt prioritization methodologies that have been suggested in both academic research and industry.
Analyzing the concept of technical debt in the context of agile software development: A systematic literature review	Behutiye, W. N. et al.	2017	The objective of this study is to examine and consolidate the current knowledge on technical debt, including its origins, impacts, and solutions for managing it within the framework of agile software development.
A systematic mapping study on technical debt and its management	Li, Z. et al.	2015	The objective of this research is to gather information on technical debt and its management, and conduct a systematic classification and thematic analysis of existing studies. This will provide a full grasp of the notion of technical debt and an overview of the current research on technical debt management.

An overview and comparison of technical debt measurement tools	Avgeriou, P. et al.	2020	Various tools employ distinct terminology, metrics, and methods to identify and quantify technical debt. The authors aim to elucidate the situation by juxtaposing the characteristics and prevalence of technical debt measurement tools and scrutinizing the available empirical data regarding their soundness.
Technical Debt Tools: A Systematic Mapping Study	Saraiva, D. et al.	2021	This study examines the present status of technical debt tools by defining the activities, functions, and types of technical debt that are addressed by existing tools designed to manage technical debt in software projects.
On Coherence in Technical Debt Research: Awareness of the Risks Stemming from the Metaphorical Origin and Relevant Remediation Strategies	Stochel, M. et al.	2020	This survey report examines the extent to which technical debt language is used consistently and aligns with the agreed-upon conceptual model in current research. The consistency of addressing technical debt is crucial for decision makers, as they may hesitate or even forgo investing in a particular aspect of the product unless the advantages of repaying the specific technical debt are sufficiently evident.
Investigate, Identify and Estimate The Technical Debt: A Systematic Mapping Study	BenIdris, M. et al.	2018	To investigate and comprehend Technical Debt (TD) in the software business, as well as have a comprehensive understanding of the present status of TD research. A total of forty-three empirical papers on TD were gathered for classification and analysis.

Preventing Technical Debt by Technical Debt Aware Project Management	Wiese, M. et al.	2021	This research assesses a strategy that prioritizes the prevention and payback of TD (technology debt). The technology was created and implemented within the information technology department of a publishing company. The distinctive feature of this framework is in the incorporation of TD management inside project management. The evaluation was conducted through a study that utilized ticket statistics and a structured survey including people from both the observed IT unit and a comparison unit.

Table 2: *Related Studies*

CHAPTER 4

TD TOOLS AND RESULTS OF EVALUATION

This section will provide concise definitions of popular technical tools used in the software business. These definitions have been obtained from a literature review within the primary categories described above. Subsequently, the evaluation of these technical instruments based on the criteria listed above will be outlined.

4.1. Tools for Technical Debt Management

Multiple tools exist for effectively controlling technical debt in the field of software development. These techniques often belong to several groups, each focusing on various aspects of technical debt.

In this part of the study, the main categories and evaluation criteria of TD tools will be mentioned.

4.1.1. The Main Categories of TD Tools

Technical debt tools typically belong to distinct groups, each designed to tackle unique facets of software development and upkeep. According to the literature review and some popular blogs like "forrester", "*medium*", "*trustradius*", "*softwareadvice*", "*peerspot*", "*stackoverflow*", "*gartner*" "*thectoclub*" and "*comparitech*". The basic classification of technical debt tools are given in Table 3 [13].

Category	Definition
Code Quality and Static Analysis Tools	These tools mostly assess the source code's quality without executing it. They detect possible problems, such as code smells, compliance with coding standards, and security vulnerabilities, by using static code analysis. The objective is to uphold the standard of code and avoid the accumulation of technical debt during the development process.

Dynamic Analysis Tools (Runtime Analysis)	Dynamic analysis tools primarily assess the runtime behavior of a software application. Dynamic analysis tools differ from static analysis tools in that they analyze the code during execution, allowing for the identification of issues like as memory leaks, performance bottlenecks, and unexpected runtime behaviors. Dynamic analysis techniques are essential in detecting and resolving technical debt associated with runtime problems. Technical debt can arise from memory leaks, inefficient algorithms, and inferior performance. Dynamic analysis techniques enable development teams to identify specific sections of the code that require runtime enhancements, thereby minimizing the technical debt associated with performance and stability.
Architectural Analysis Tools	Architectural analysis tools primarily assess the architecture and structure of software systems. They assist in identifying architectural challenges and probable design faults that could contribute to the accumulation of technical debt. These techniques assist in preserving a resilient and expandable architecture over a period of time.
Visualization Tools	Visualization tools aid developers in comprehending and examining codebases, dependencies, and other software-related structures by means of graphical representations. These technologies frequently utilize charts, diagrams, and graphs to effectively communicate intricate information, hence enhancing its accessibility for developers and stakeholders.

Automated Testing Tools	Automated testing tools facilitate the detection of areas lacking sufficient or comprehensive test coverage. The mentioned components encompass unit testing frameworks, UI testing tools, and other testing suites. The objective is to guarantee comprehensive test coverage, minimizing the possibility of incurring technical debt when making code modifications.
Dependency Analysis Tools	Dependency analysis tools assist teams in effectively managing and comprehending the interdependencies present within a codebase. They ascertain the interdependencies among components, libraries, and modules, resolving technical obligations associated with obsolete or troublesome dependencies, thereby assuring a more robust and sustainable system.
Security Analysis Tools	Security analysis tools are purposefully created to detect and resolve security weaknesses in a codebase. Their role involves conducting both static and dynamic analysis to identify potential risks, assisting teams in addressing technical debt related to security faults and vulnerabilities.
Continuous Integration Tools	Continuous Integration tools automate the process of regularly integrating code changes from multiple contributors into a shared repository. They help ensure that new code integrates smoothly with the existing codebase and that automated tests are run to catch potential issues early in the development lifecycle.
Repository and Project Management Tools	Repository and project management tools help teams collaborate, track changes, and organize their work. They provide features such as version control, issue tracking, project planning, and documentation, making

	it easier for development teams to coordinate efforts and manage the development lifecycle.
Documentation Tools	Documentation tools facilitate the creation, organization, and upkeep of documentation for codebases. These tools encompass features for producing API documentation, code comments, and comprehensive project documentation. Thorough documentation mitigates knowledge transfer challenges and tackles technical debt associated with comprehending and maintaining code.
User Feedback and Bug Tracking Tools	Tools in this category streamline the process of gathering and organizing user feedback and issue reports. These systems encompass problem tracking capabilities that assist teams in prioritizing and resolving reported issues. By rapidly addressing problems raised by users, these technologies help to reduce technological debt associated with user experience and system stability.

Table 3: *Main Categories and Definitions of Technical Debt Tools*

4.1.2. Main Evaluation Criteria for TD Tools

Choosing the appropriate tools for your software development process is essential for the successful completion of a project. After reviewing academic studies conducted by Lenarduzzi, V., et al. (2021) and Pavlič, L., et al. (2019) and some popular websites mentioned in the previous section, it is important to consider the following primary factors given in Table 4 to compare tools in different categories.

Criteria	Definition
Functionality	Does the tool respond to the particular requirements of your team and project? Make sure that it offers extensive capability in the key areas of your development process, including code quality, testing, monitoring, documentation, visualization, and project management.
Ease of Use	Does the tool's user interface include an intuitive and user-friendly design? An intuitively navigable and user-friendly tool can enhance the adoption and efficiency of team members.
Integration Capabilities	*Compatibility:* Does the product have the capability to smoothly incorporate into your current development ecosystem, encompassing version control systems, issue tracking, and continuous integration pipelines? *API Support:* Does the application have APIs or connectors that enable customization and integration with other technologies utilized by your team?
Customization Options	*Configurability:* Is it possible to tailor the tool to conform to your team's procedures and coding standards? Scalability: Does the tool exhibit efficient scalability as the complexity and size of your project increase?
Scalability and Performance	*Performance:* What is the tool's performance in terms of speed and responsiveness, particularly when dealing with larger codebases and projects? *Resource Requirements:* Take into account the resource demands of the tool, encompassing memory utilization and computational capacity.
Community and Support	*Community Engagement:* Does the tool have a vibrant and engaged community? An active community

	frequently entails enhanced assistance, regular updates, and an abundance of resources. *Support Options:* What amount of assistance does the tool's seller or community provide? Take into account variables such as manuals, forums, and customer service channels.
Cost and Licensing	*Licensing Model:* Comprehend the licensing framework of the tool. Make sure it conforms to your financial constraints and project specifications. *Total Cost of Ownership:* Take into account the total expenditure, encompassing licensing fees, maintenance costs, and any training expenditures.
Security and Compliance	*Security Features:* Does the tool have robust security features to protect sensitive data and code repositories? *Compliance:* Ensure the tool complies with relevant industry standards and regulations if applicable.
Flexibility and Extensibility	*Plugin Ecosystem:* Does the tool have the capability to accommodate plugins or extensions? This can improve its functionality and flexibility to meet changing requirements. *Customization Capabilities:* Assess the tool's capacity to be tailored and expanded to meet specific project needs.

Table 4: *Main Criteria and Definitions for Evaluating Technical Debt Tools*

4.2. TD Tools in Code Quality, Static Code and Security Analysis Category

Static code analysis, sometimes referred to as Static Application Security Testing (SAST), involves the examination of computer program without executing the software. Developers employ static code analysis tools to identify and rectify vulnerabilities, defects, and security issues in their newly developed applications during the static phase of the source code, which refers to the period when it is not being executed.

According to the literature reviews and some popular blogs such as "forrester", "*medium*", "*trustradius*", "*softwareadvice*", "*peerspot*", "*stackoverflow*", "*gartner*" "*thectoclub*" and "*comparitech*" etc. some popular tools for code quality and static code analysis can be seen in Table 5.

Tool Name	Brief Definition
SonarQube	A platform for continuous inspection of code quality that detects bugs, security vulnerabilities, and code smells.
Checkmarx	A static application security testing (SAST) tool that identifies security vulnerabilities in the source code.
Synopsys Coverity	Synopsys Coverity is a static code analysis tool designed to assist DevOps teams in promptly identifying and resolving security vulnerabilities during the software development process. The system provides both cloud-based and on-premise deployment alternatives.
ReSharper	ReSharper is a plugin designed for Visual Studio, which is an integrated development environment (IDE) used for the Microsoft .NET Platform. The tool is capable of conducting code quality analysis for programming languages such as VB.NET, JavaScript, HTML, CSS, and XML.
CAST	CAST Highlight is a software intelligence platform capable of analyzing the source code of numerous applications. The software produces informative dashboards with color-coded visuals that offer quick and comprehensive insights into your applications.
CodeClimate	Code Climate Quality is a software application that does code analysis to assist development teams in delivering higher quality code. The

	tool offers static analysis capabilities for programming languages such as PHP, Java, JavaScript, Python, and Ruby.
Snyk Code	Snyk is a developer security platform that provides immediate scanning and analysis for your code. Additionally, it provides git repository integration, enabling you to prioritize bugs across all of your projects.
Micro Focus Fortify Static Code Analyzer (SCA)	The tool does static code analysis to identify the underlying causes of vulnerabilities, categorizes issues based on their severity, and offers comprehensive remediation guides. Additionally, it has dynamic application testing and source code analysis capabilities.
Codacy	Another exceptional option within the realm of static analysis tools that assists in evaluating the quality of our code. It obstructs the merging of pull requests that do not meet your quality standards and aids in averting significant problems from impacting your product.
PVS Studio	PVS Studio is well known for its proficiency in identifying software defects and vulnerabilities. It provides a digital compendium of analytic rules and analysis codes for errors, dead snippets, typos, and repetition.

Table 5: *TD Tools for Code Quality and Static Code Analysis*

4.3. Evaluation of TD Tools in Code Quality and Static Code Analysis Category

According to the literature reviews about the tools given at Table 5 and approximately 100 reviews written by users; main features, strong and weak points of these tools are given in Table 6.

Tool Name	Evaluation Results
SonarQube [33]	Pros: • It is an open-source platform • It can be self-hosted or deployed to the cloud • The Community Edition is highly comprehensive, encompassing security analysis and bug detection, making it particularly well-suited for development environments

	• Supports over 30+ programming languages, including Java, Ruby, and C • Offers integrations with popular DevOps platforms • Performs continuous code inspections • The system has the ability to classify each infraction according to its severity, ranging from minor to significant, and also provides an estimate of the required time to address the issue • Users can create "quality gates" to control that new code must exceed this gate value Cons: • May produce false positives • Free version has limited functionality OS: Docker over Windows, macOS, Linux, and Azure Pricing: SonarQube is priced per instance per year and based on your lines of code. The price starts: • For developer $150 /year/100K LOC • For developer $20000 /year/1M LOC Trial: 14-day free trial Official Sites: https://www.sonarsource.com/
Checkmarx [44]	Pros: • Its product is an enterprise-grade, flexible, and accurate static analysis tool • Best Fix Location: This capability enables you to pinpoint the optimal location for fixing a single line of code and address numerous issues simultaneously • Tailored App Protection: With over 40 presets and the ability to create custom queries, you may tailor SAST to suit the specific requirements of every application and business objective • AI Query Builder: AI Query Builder generates new, and customizes existing, queries to better tailor searche • Checkmarx SAST is an integral component of an automated testing platform that also encompasses dynamic testing techniques, allowing for their seamless integration. The tool can be integrated with code repositories and bug trackers, allowing the tester to be automatically launched as part of the code submission process

	• Checkmarx SAST conducts static application security testing (SAST) scans immediately upon code check-in, directly from source code repositories such as GitHub, GitLab, Azure, and Bitbucket. This enables seamless integration into your software development life cycle (SDLC) • Checkmarx SAST is compatible with more than 50 programming languages and 80 language frameworks. It can handle both the latest and older languages, making it suitable for multi-platform development <u>Cons:</u> • No free trial version • No price information <u>Official Sites</u>: https://checkmarx.com/? ** Checkmarx is a cloud-based SaaS package, so, those who want a hosted application testing package instead of one that needs to be self-managed would prefer Checkmarx over SonarQube.
Synopsys Coverity [45]	<u>Pros:</u> • Real-time detection helps deal with errors quickly • Able to scan lines of code quicker than other tools • Provides detailed reports • The Code Sight IDE plugin enables developers to identify and rectify security or quality concerns in real-time while writing their code • The system demonstrates exceptional precision in detecting vulnerabilities like as buffer overflows, input validation issues, and memory leaks <u>Cons:</u> • Complicated to integrate with other tools • User interface is difficult to navigate • No price information <u>Pricing</u>: Pricing upon request <u>Trial</u>: Trial license available <u>Official Sites</u>: https://www.synopsys.com/ ** Synopsys is primarily designed for utilization inside the development aspect of DevOps, rather than being utilized by operations teams. This program rivals the self-hosted SonarQube as it is compatible with Windows, macOS, and Linux operating

	systems for installation. In addition, it rivals Checkmarx as it offers subscription-based services through the Synopsys SaaS platform.
ReSharper [31]	Pros: • Offers tight integration with Visual Studio • Has extensive documentation to help you learn the tool • Provides a helpful auto-complete list that appears as you code • It provides a comprehensive range of refactoring capabilities that allow you to modify your code base securely • It promptly identifies coding problems and includes more than a thousand immediate solutions. To rectify any highlighted issue, simply hit the "Alt+Enter" key combination Cons: • Requires a paid license to use • Large code base can slow down Visual Studio Pricing: From $349.0/user/year Trial: 30-day free trial Official Sites: https://www.jetbrains.com/resharper/
CAST [46]	Pros: • Best for performing software assessments at scale • Offers cloud migration suggestions • Supports over 40 programming languages • It produces informative dashboards with color-coded visuals, allowing you quick and comprehensive insights into your applications • The tool performs local code scans and never uploads your code to the cloud • Integrations are available natively for GitHub, Bitbucket, and Azure DevOps. You can also use CAST Highlight's public REST API to extract and integrate key metrics into other systems Cons: • Requires a paid license to use • Large code base can slow down Visual Studio Pricing: Single App/1 Application On boarded/$6,000/Year

	Trial: 30-day free trial Official Sites: https://www.castsoftware.com/
CodeClimate [31]	Pros: • Suitable for GitHub users, it offers two-factor authentication with GitHub OAuth • Provides static analysis for languages like PHP, Java, JavaScript, Python, and Ruby • It also provides a concise summary of any problems with a pull request prior to merging it into the primary repository. The GitHub browser add-on is useful for presenting test coverage data on a line-by-line basis • Provides visual progress reports with a simple grading system • The tool also integrates natively with ticket and messaging systems like Asana, Trello, and Slack Cons: • May generate false positives • Free plan has limited functionality Pricing: From $16.67 per month Trial: Free for open-source projects Official Sites: https://codeclimate.com/
Snyk Code	Pros: • Developer security platform that offers real-time scanning and analysis • It also offers GIT repository integration • It's DeepCode AI tool pulls up a list of quick fixes as it identifies issues • It assigns a risk score to each issue, enabling you to prioritize them • Easy to integrate and setup • Snyk is the ideal tool for businesses and developers who prefer the cloud computing environment - it can find and fix vulnerabilities in code, containers, Kubernetes, and Terraform • Integrations are available natively for CI/CD tools like Jenkins, Azure Pipelines, and Bitbucket Pipelines. There are also plugins for IDE tools like Eclipse, PhpStorm, and Visual Studio • Snyk provides actionable fix advice in your tools. With auto PRs

	Cons: • Slower scan times • No self-hosted option • Free plan limited to 100 tests per month Pricing: Starting at $25 per month/product Trial: Free plan available Official Sites: https://snyk.io/
Micro Focus Fortify Static Code Analyzer (SCA)	Pros: • The user interface is intuitive, and the dashboard is valuable for monitoring any identified issues. • Offers compatibility with many programming languages and frameworks • Wide variety of integrations accessible • This tool offers dynamic (DAST) application testing as well as source code analysis (SAST). • It can be integrated into IDEs like Eclipse or Visual Studio • The tool offers unlimited flexibility with its multiple deployment modes Fortify SAST offers options for on-premises, SaaS, or hybrid methods Cons: • Can be difficult to set up initially • Not able to deal with false positive detection well Pricing: Pricing upon request Trial: No free trial Official Sites: https://www.microfocus.com/
Codacy	Pros: • Best for continuous integration (CI) workflows • The platform supports over 40 programming languages and frameworks • Integrating Codacy with GitHub allows to get instant feedback on code • It helps standardize code quality by automatically blocking pull requests that don't meet certain standards • Ability to set custom rule sets, also upload your own configuration file • Adheres to SOC2 security standards • Integrations are available natively with GitHub, GitLab, and Bitbucket

	• Native integrations are also available for Jira and Slack Cons: • Doesn't integrate with Lombok, a Java library that reduces boilerplate code • Not able to export code patterns Pricing: Open-Source Edition $0 and PRO Edition $15 Per developer/month billed annually or $18 billed monthly Trial: 14-day free trial Official Sites: https://www.codacy.com/
PVS Studio	Pros: • Best for game developers • PVS-Studio is a code analyzer that can detect bugs and security flaws in source code written in C, C++, C#, and Java • It offers direct integrations with Unity and Unreal Engine • Integrations are available natively for over 30 platforms, including Visual Studio, Maven, Jenkins, Docker, and Azure DevOps • Integrates with bug tracking systems like GitHub Issue • Offers extensive documentation • Works on multiple operating systems, like Windows, macOS, and Linux Cons: • Only supports a small number of programming languages • Can use up a lot of resources for large code bases Pricing: Pricing upon request Trial: 7-day free trial Official Sites: https://pvs-studio.com/en/

Table 6: *Evaluation of TD Tools for Code Quality and Static Code Analysis*

4.4. TD Tools in Dynamic Analysis Category (DAST Tools)

Dynamic Application Security Testing (DAST) solutions employ simulated assaults or penetration tests to detect real-time vulnerabilities in online applications that are currently operational. They consistently analyze apps for potential vulnerabilities that could be exploited by cybercriminals through attacks like as SQL injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF), among other methods. After identifying a vulnerability, the DAST tool promptly notifies the development team, enabling them to promptly address and resolve the issue.

According to the literature reviews and popular blogs searched in this study, some popular tools for dynamic analysis can be seen in Table 7.

Tool Name	Brief Definition
Intruder	Intruder is a cloud-native vulnerability management software that facilitates security monitoring, risk assessment, configuration mapping, and bug detection.
SOOS DAST	SOOS DAST seamlessly integrates into the build workflow and combines DAST test findings with SCA vulnerability checks in a unified and robust online dashboard.
Invicti	Invicti, previously known as Netsparker, is an interactive application security testing package (IAST) that incorporates DAST procedures. The plans for this tool include features that make it well-suited for usage as a vulnerability scanner, an automated pen testing tool, and a continuous testing system.

Table 7: *TD Tools for Dynamic Analysis*

4.5. Evaluation of TD Tools in Dynamic Analysis Category

According to the literature reviews about the tools given at Table 7 and approximately 80 reviews written by users; main features, strong and weak points of these tools are given in Table 8.

Tool Name	Evaluation Results
Intruder [41]	Intruder is an automated and dynamic vulnerability management solution that operates in the cloud. It effortlessly conducts scans on infrastructure, online applications, and APIs. It provides practical and situation-specific outcomes, allowing users to prioritize the most crucial security concerns initially. Intruder offers continuous protection by conducting regular vulnerability

	checks and actively monitoring for emerging threats, effectively minimizing the potential for attacks on the system. Pros: • Integration with development project tools • ServiceNow integration for operations support • Attack surface management • Its testing services are priced per instance, no need to pay for that are not used • Integrates with code repositories • An easy-to-use Web-based console • Risk scoring • Continuous scanning made simple. Proactive protection from emerging dangers. Business context is used to prioritize intelligent results Cons: • Price • DAST is not part of the core package of any edition Pricing: • 14 day free trial is available • Essential version $157 for one app/per month • Pro version $221 for one app/per month • Premium version $3633 for one app/per year Official Sites: https://www.intruder.io/
SOOS DAST [42]	SOOS is a self-governing software security firm situated in Winooski, VT USA. We specialize in developing security software specifically designed for your team. SOOS: Streamlined approach to software security. Utilize the SOOS Core SCA tool to do a thorough examination of your software, identifying any vulnerabilities and potential open source license complications. Pros: • HTML App DAST Tests & Single Page App DAST Tests

	• REST API & SOAP Testing & GraphQL Testing • Open Source License Management • Script Configurations & Easy Setup • Role-Based Dashboard for Engineering/Legal/Security Viewers Cons: • Learning curve and price Pricing: • Free trial is available • $100 for 5 developers / per month Official Sites: https://soos.io/
Invicti [43]	Intruder is an automated and dynamic vulnerability management solution that operates in the cloud. It effortlessly conducts scans on infrastructure, online applications, and APIs. It provides practical and situation-specific outcomes, allowing users to prioritize the most crucial security concerns initially. Intruder offers continuous protection by conducting regular vulnerability checks and actively monitoring for emerging threats, effectively minimizing the potential for attacks on the system. Pros: • Users can use this system on-demand or on a schedule to check the security of live systems or set it up within CI/CD pipeline framework as a continuous tester. This is an IAST system, but it implements DAST procedures as well. • Cloud-based or on-premises • Continuous testing • Vulnerability scanning option • Suitable for development testing • Installs on Windows and Windows Server • Highly visual interface Cons: • It is an advanced security tool for professionals, not ideal for home users

	Pricing: • Trial version is available • Premium and enterprise edition no price info Official Sites: https://www.invicti.com/

Table 8: *Evaluation of TD Tools for Dynamic Analysis*

4.6. TD Tools in Architectural and Dependency Analysis Category

Architectural analysis tools are specialized software tools created to aid in the assessment, design, and enhancement of the structure of a software system. These tools offer valuable insights into several facets of the system's architecture, enabling developers and architects to make well-informed decisions on design, performance, and maintainability.

According to the literature reviews and popular blogs searched in this study, some popular tools for architectural and dependency analysis can be seen in Table 9 [28].

Tool Name	Brief Definition
Structure101	Structure101 offers graphical depictions of codebases, aiding teams in comprehending and overseeing intricate software architectures. It provides a visual representation of interdependencies, ensures compliance with architectural guidelines, and highlights potential areas for enhancement.
Sonargraph	Sonargraph is a software application that provides architectural visualization, analytics, and dependency analysis for managing software architecture and quality. It facilitates the identification and resolution of issues pertaining to the code's structure and design.
JArchitect	JArchitect is a Java static analysis tool that offers valuable information on code quality, design, and architecture. The tool provides visual representations, quantifiable measurements, and trend evaluations to assist teams in upholding a robust codebase.
SonarQube (Architecture Plugin)	SonarQube, renowned for its static code analysis capabilities, also provides architecture analysis through the use of plugins. It

	offers visual representations and measurements to evaluate the condition of the codebase and compliance with architectural principles.
NDepend	NDepend is a software tool that performs static analysis on programs built using the .NET framework. It provides valuable information on the quality, design, and structure of code, assisting teams in visualizing interdependencies, identifying problematic code patterns, and effectively managing technical debt.

Table 9: *TD Tools for Architectural Analysis*

4.7. Evaluation of TD Tools in Architectural and Dependency Analysis Category

According to the literature reviews about the tools given at Table 9 and approximately 150 reviews written by users; main features, strong and weak points of these tools are given in Table 10.

Tool Name	Brief Definition
Structure101 [39]	Structure101 is utilized to visually represent the architecture of an application using a graph that displays the relationships between modules, packages, and classes, or by a presentation of a dependency matrix. Additionally, it is beneficial to construct the architecture diagram while incorporating support for violation checks. This allows for the identification of tangles, fat packages, classes, and methods inside the code-base. Pros: • Simulate Restructuring • Create Task-Specific Views: Tag the dependencies of an item, isolate the tagged items (filtering), hide packaging (slicing), expand all to show a complete call graph, isolate further for paths between 2 items, show results with packaging or without. • Organize Modules Into Groups

	• Constrain Module Dependencies • Create Dependency Validation Diagrams • Use Model Views To Analyze Structure • See How Structure Changes Over Time • Available for C/C++, Java, .Net, & more Cons: • Learning curve • Price Pricing: starts from $349.00 per user/year Official Sites: https://structure101.com/
Sonargraph-Architect [33]	Sonargraph-Architect calculates several code and quality metrics that can be utilized to promptly evaluate the technical quality of any software system. Pros: • It supports C#, C/C++, Java/Kotlin and Python 3 • The software offers robust visualization views, numerous metrics, automated architecture checks using a powerful DSL, a Groovy-based scripting engine, a duplicate code checker, virtual refactorings, an issue resolution workflow, advanced metrics such as LCOM4, and a computer for breaking up cyclic dependencies • Users can define quality gates based many different criteria and these gates can be configured to break the build if things got worse in comparison to the baseline • Sonargraph has very powerful dependency visualization features • Design architecture using Sonargraph's Architecture DSL • Virtual Refactorings allow the Simulation of Refactorings without touching the Code • Break up cyclic dependencies with minimal Effort • Create Your Own Code Checkers Cons: • Price

	• Learning curve Pricing: • 14 day free trial is avaliable • Pricing changes over license type ex: Java $360.00 per/month Official Sites: https://www.hello2morrow.com/products/sonargraph
JArchitect [40]	JArchitect is a prominent tool in the field of static code analysis for Java. It stands out for its ability to visually represent the architecture of Java code. Complexity in programming projects manifests as an organized and graphical representation, highlighting intricacies in the source code. Pros: • Proficient at representing intricate Java code structures. • Offers invaluable insights into technical debt and code smells. • Excellent integration capabilities, especially with GitHub and Jenkins. • Smart Technical-Debt Estimation, Fast Estimation, Customizable • Detect Dependency Cycle • Lots of default Quality Gates are proposed by JArchitect • Within seconds, users can determine the specific portion of the code that will be affected by refactoring a class. Additionally, users will receive notifications if there is an unintentional violation of layer dependencies. Furthermore, users can precisely identify the section of the code that relies on a specific tier component, as well as generate a list of methods that can be accessed from a given method Cons: • Some users might find the interface slightly daunting initially. • Certain functionalities may appear redundant for basic projects. • Price • Requires a fair bit of configuration for optimal results.

	<u>Pricing</u>: • JArchitect for Developer $ 599 per user/year • JArchitect DevOps License $ 3999 per user/year <u>Official Sites</u>: https://www.jarchitect.com/
NDepend [38]	JArchitect is a prominent tool in the field of static code analysis for Java. It stands out for its ability to visually represent the architecture of Java code. Complexity in programming projects manifests as an organized and graphical representation, highlighting intricacies in the source code. <u>Pros:</u> • Code Rule and Code Query: There are numerous default code standards that can be used to evaluate adherence to best practices. Code Query over C# LINQ (CQLinq) is supported to facilitate the customization of rules for code querying • Powerful Dependency Graph and Matrix • Smart Technical Debt Estimation • Quality Gates: Quality Gates are C# LINQ (CQLinq) queries that implement PASS/FAIL criteria to code quality • In-Depth Issues Management • Complexity and Diagrams: Identify intricate code with ease using exceptional diagramming features • Detect Dependency Cycles • NDepend.API and Power Tools: Write your own static analyzer based on NDepend.API, or tweak existing open-sources Power Tools <u>Cons:</u> • For .NET projects • Price <u>Pricing</u>: • NDepend v2023.2.3 for Developer $ 492 per user/year • Free trial is available <u>Official Sites</u>: https://www.ndepend.com/

Table 10: *Evaluation of TD Tools for Architectural and Dependency Analysis*

4.8. TD Tools in Automated Testing Category

Automated testing solutions facilitate the reduction of testing durations, augmentation of test scope and speed of execution, and guarantee the efficient utilization of test cases with minimal human involvement.

According to the literature reviews and popular blogs searched in this study, some popular tools for automated testing can be seen in Table 11.

Tool Name	Brief Definition
LambdaTest	LambdaTest offers cloud-based automated testing services. The cloud solution enables teams to expand their test coverage by conducting rapid parallel testing across several browsers and devices.
Selenium	Selenium is a freely available framework used for automating web browsers. It is extensively employed to test web applications by imitating user activities, aiding in verifying the functionality and efficiency of web-based systems.
TestComplete	TestComplete is a paid automated testing solution that provides support for several forms of testing, such as functional, regression, and UI testing. It offers assistance for conducting application testing on many platforms and in varied scenarios.
Appium	Appium is a freely available framework for automating mobile applications on both iOS and Android platforms. It is widely favored for mobile testing due to its compatibility with native, hybrid, and mobile web applications.
Cypress	Cypress is a JavaScript framework for doing end-to-end testing. It offers a testing environment that is efficient, dependable, and user-friendly. Its purpose is to streamline the process of testing web applications.

Table 11: *TD Tools for Automated Testing*

4.9. Evaluation of TD Tools in Automated Testing Category

According to the literature reviews about the tools given at Table 11 and approximately 50 reviews written by users; main features, strong and weak points of these tools are given in Table 12.

Tool Name	Brief Definition
LambdaTest [36]	The LambdaTest framework guarantees extensive browser coverage through its cross-platform interoperability with over 40 browsers, ensuring reliable and accurate results regardless of the operating system being used. The platform can be utilized with virtual devices hosted on the cloud or emulators deployed locally. Pros: • It allows to test web applications across a wide range of browsers and operating systems. • It grants users access to real web browsers operating on actual machines, hence providing a more precise testing environment. • The platform supports parallel testing that speeds up test execution. • LambdaTest integrates with popular testing frameworks and CI/CD tools. • It provides AI-powered test analytics and smart TV testing • It provides responsive testing to ensure applications work well on various devices and screen sizes. Cons: • New users of the platform may encounter a period of adjustment when it comes to establishing and customizing tests • Some configurations may be time-consuming • Some users may perceive that its analytics and reporting lack sufficient detail or customization options Pricing: From $15/month Trial: Free trial available Official Sites: https://www.lambdatest.com/

Selenium [29]	Selenium is a dependable option for test automation, well regarded for its open-source characteristics. Through thorough testing and analysis, it is evident that Selenium is intricately crafted for the purpose of automating web browsers. <u>Pros:</u> • Runs tests across different browsers • Supports various operating systems • Executes tests while the browser is minimized • It is free • It allows testing across multiple web browsers • Supports Windows, macOS, and Linux, enabling cross-platform testing • Provides support for multiple programming languages • Extensive user community, resulting in a wealth of online resources, lessons, and community assistance • It provides flexibility in scripting and is capable of managing intricate testing scenarios • Allows for parallel test execution • Integrates with various testing frameworks and continuous integration tools like Jenkins <u>Cons:</u> • Selenium scripts may require frequent updates, especially when web application UI changes • Limited Support for Non-Web Applications not suitable for automating desktop or mobile applications • Selenium's speed can be relatively slower compared to some commercial automated testing tools • Setting up and managing parallel testing environments can be complex and require additional configurations • May struggle with handling dynamic elements <u>Official Sites</u>: https://www.selenium.dev/

TestComplete [32]	TestComplete is a graphical user interface (GUI) test automation tool provided by Smartbear. It is designed to support a diverse variety of applications, such as desktop, web, and mobile, and can be used by anyone with different levels of technical knowledge. Pros: • supports testing for various types of applications, including desktop, web, and mobile • Powerful Cloud-Based Testing • Superior Object Recognition • Enterprise Application Support: compatible with testing enterprise-level applications such as SAP, Oracle EBS, and Salesforce • Allows for the execution of functional UI tests in parallel • Integrates with other tools in the software development ecosystem, including CI/CD pipelines, test management systems, issue tracking, and version control Cons: • Although the interface is designed to be easy to use, there may still be a period of time required to learn how to use more advanced features and scripting • TestComplete necessitates continuous maintenance to ensure that test scripts remain current with any modifications made to the application. • TestComplete primarily targets Windows environments • The cost of TestComplete may be a consideration for smaller organizations or individual users Pricing: • trial version is available • TestComplete Base - Starting at €3,253 • TestComplete Pro - €5,045 Official Sites: https://smartbear.com/product/testcomplete/

Appium [32]	Unlike other software testing approaches, Appium focuses on automating UI tests. This allows testers to write code that directly interacts with the application's user interface, following certain user scenarios. Pros: • Mobile-focused test automation features • Remote testing capabilities for large, distributed teams • Customizable insight generation • Appium offers cross-platform compatibility, allowing for the automation of mobile applications on several platforms such as Android and iOS using a unified codebase. This minimizes the time and exertion required to create distinct test scripts for each platform. • Extensive Language Compatibility: Appium offers support for a wide range of programming languages such as Java, Python, Ruby, JavaScript, and others, enabling testers to utilize their choice scripting language. • Appium allows for testing on both physical devices and virtual emulators/simulators, offering versatility in the testing environment. Cons: • Complex Setup • Slower Execution • Limited Built-In Reporting • Steep Learning Curve Pricing: • Free and open-source Official Sites: https://appium.io/docs/en/2.3/
Cypress [37]	Cypress specializes in comprehensive testing, particularly for applications that utilize contemporary JavaScript frameworks. Cypress is highly compatible with projects developed using contemporary frameworks such as Vue, Angular, and React.

	Pros: • Real-time execution of commands with visual feedback • Automatically wait for assertions and commands • Captures screenshots during test execution • Simple installation process. It requires no complex setup or additional dependencies • Real-Time Support: Test can be written while the application is being built, allowing for immediate feedback and agile testing • Instant Test Execution • Comprehensive Test Snapshots: simplifies the debugging process by providing test snapshots right from the command log • Documentation, Useful JavaScript Tools Cons: • Limited Browser Compatibility • Lack of Support for Iframes • Learning Curve Pricing: • Cypress offers a free package with three users and 500 test results, as well as three paid packages for teams, businesses, and enterprises. Official Sites: https://www.cypress.io/

Table 12: *Evaluation of TD Tools for Automated Testing*

4.10. TD Tools in Continuous Integration Category

Continuous Integration (CI) is a software development approach that involves the frequent and automated integration of code changes into a shared repository. The main objective of Continuous Integration (CI) is to identify and resolve integration problems at an early stage in the development process, resulting in a more efficient and smooth development workflow.

According to the literature reviews and popular blogs searched in this study, some popular tools for continuous integration can be seen in Table 13 [27].

Tool Name	Brief Definition
Jenkins	Jenkins is a freely available automation server that provides support for the construction, deployment, and automation of various projects. It has a high degree of extensibility through the use of plugins and is commonly employed for the implementation of continuous integration and continuous delivery (CI/CD) pipelines.
GitLab CI/CD	GitLab CI/CD is an intrinsic component of the GitLab platform, offering an embedded CI/CD system. Developers can define, test, and automate the construction and deployment of their projects right within the GitLab environment.
Travis CI	Travis CI is a cloud-based Continuous Integration (CI) solution that seamlessly connects with repositories hosted on GitHub. It autonomously constructs and evaluates modifications to the code, aiding developers in promptly detecting problems throughout the development phase.
CircleCI	CircleCI is a cloud-hosted Continuous Integration/Continuous Deployment (CI/CD) technology that streamlines the software development process. The platform facilitates the construction, evaluation, and implementation of apps and seamlessly interfaces with widely used version control systems like GitHub and Bitbucket.
AWS CodePipeline	AWS CodePipeline is an entirely supervised service for continuous integration and continuous delivery (CI/CD) offered by Amazon Web Services (AWS). Although AWS CodePipeline may not consistently appear in compilations of the most popular CI/CD technologies, it is a notable and extensively utilized service inside the AWS ecosystem. The popularity of this service among enterprises utilizing AWS infrastructure is due to its smooth interoperability with multiple deployment scenarios and its interaction with other AWS services.

Table 13: *TD Tools for Continuous Integration*

4.11. Evaluation of TD Tools in Continuous Integration Category

According to the literature reviews about the tools given at Table-12 and approximately 70 reviews written by users; main features, strong and weak points of these tools are given in Table 14.

Tool Name	Brief Definition
Jenkins [34]	Jenkins is a freely available automation server that serves as the central hub for executing build and continuous integration tasks. The program is written in Java and includes packages for Windows, macOS, and Unix-like operating systems. Jenkins has extensive plugin support, enabling the creation, deploying, and automation of software development projects. Pros: • High customizability and flexibility, making it suitable for diverse project needs. • Active community support and continuous development ensure its continued improvement. • Suitable for small- and large-scale projects due to its distributed build capabilities. • Best for customizable build pipelines Cons: • For novices, Jenkins might pose a more challenging learning experience, particularly when navigating intricate setups and plugins. • Requires careful management and can be resource-intensive on larger projects Pricing: Free Official Sites: https://www.jenkins.io/
	GitLab is a comprehensive set of tools designed to effectively manage various areas of the software development lifecycle. Users can initiate builds, execute tests, and deploy code with every revision or push. Furthermore, customers have the

GitLab CI/CD [27]	capability to construct jobs within a virtual machine, Docker container, or on an alternative server. Pros: • Access, generate, and oversee codes and project data using branching tools • Utilize a centralized distributed version control system to efficiently create, enhance, and oversee codes and project data, facilitating quick iteration and delivery of business values. • Offers a centralized and expandable platform for collaborating on projects and code, ensuring accuracy and scalability. • Facilitates the complete adoption of Continuous Integration (CI) by automating the processes of building, integrating, and verifying source codes for delivery teams. • Offers container scanning, static application security testing (SAST), dynamic application security testing (DAST), and dependency scanning to ensure the development of safe applications while also ensuring compliance with licensing requirements. • Facilitates the automation and streamlining of application releases and delivery. Cons: • For novices, Jenkins might pose a more challenging learning experience, particularly when navigating intricate setups and plugins. • Requires careful management and can be resource-intensive on larger projects Pricing: Free Official Sites: https://about.gitlab.com/
	Travis CI is a cloud-based tool for continuous integration that smoothly interacts with repositories on GitHub. It initiates builds and tests automatically when there are changes in the code, pull requests, or other events. Pros:

Travis CI [27]	• Developers may rapidly enable continuous integration (CI) for their applications by utilizing the straightforward setup and configuration options provided by YAML files. • Rapid and uncomplicated installation conserves time and diminishes intricacy. • The provision of a free tier specifically for open-source projects serves to foster and promote community participation and cooperation. • Comprehensive documentation guarantees that customers may effortlessly locate resources and receive assistance. • Seamlessly incorporated with GitHub, automatically initiating builds and tests whenever there are modifications to the code or pull requests. This integration simplifies the continuous integration process for projects hosted on GitHub. • Rapid and effortless configuration for repositories hosted on GitHub. Developers may easily setup Travis CI for their projects with little configuration. • Enables matrix builds, enabling developers to do identical builds on many configurations and settings, facilitating compatibility testing across diverse setups. Cons: • The free tier imposes restrictions on concurrency and build minutes, rendering it unsuitable for bigger or resource-intensive projects • Full functionality of private repositories may necessitate upgrading to a subscription plan due to the limited support provided in the free tier Pricing: • Travis CI Enterprise Pricing - $34 Per User/Month • Open-source projects may be applied at no charge on travis-ci.org Official Sites: https://www.travis-ci.com/

CircleCI [27]	CircleCI is a Continuous Integration (CI) solution that seamlessly integrates with Github, a widely used cloud hosting platform for version control systems. CircleCi is highly versatile as it can accommodate a wide range of version control systems, container systems, and delivery techniques. CircleCi can be deployed on-premise or accessed via a cloud-based service. Pros: • Notification triggers from CI events • Performance optimized for quick builds • Easy debugging through SSH and local builds • Analytics to measure build performance • Wide array of integrations with popular tools and platforms • Support for Docker and parallel execution Cons: • May require some learning curve for newcomers • Configuration might be complex for some users • Could be over-featured for very small or simple projects • The free tier has some limitations, such as fewer build containers, which could affect the scalability of large projects. • Cost can increase for resource-intensive builds or if higher concurrency is required. Pricing: • CircleCI's pricing starts from $15/user/month. Official Sites: https://www.travis-ci.com/
AWS CodePipeline [35]	Amazon Web Services is a highly influential provider of cloud infrastructure in the market. They provide tools and services for various infrastructure and code development needs. CodePipeline is the Continuous Integration (CI) Tool provided by the company. CodePipeline has the capability to immediately integrate with various pre-existing AWS technologies, ensuring a smooth and uninterrupted AWS user experience.

	Pros: • Fully cloud • Integrated with Amazon Web services • Custom plugin support • Robust access control • Enables developers to specify personalized pipeline stages and actions. This adaptability allows for the development of advanced CI/CD procedures customized to meet unique project needs. • Enables immediate monitoring and notifications for pipeline executions, facilitating the tracking of CI/CD process progress and rapid response to any concerns • Cons: • Advanced configurations and complex setups might require a deeper understanding of AWS services and architecture. • Cost may vary depending on the number of pipeline executions and the services used in the CI/CD process. Pricing: • CircleCI's pricing starts from $15/user/month. Official Sites: https://aws.amazon.com/codepipeline/

Table 14: *Evaluation of TD Tools for Continuous Integration*

4.12. TD Tools in Repository and Project Management Category

A repository, sometimes known as a repo, is a centralized storage facility designed for storing and organizing all the information and resources related to a project. Any stakeholder or developer involved in the project has the ability to retrieve the code or resources from your repository in order to implement new features or correct bugs in the product or software application.

According to the literature reviews and popular blogs searched in this study, some popular tools for architectural and dependency analysis can be seen in Table 15.

Tool Name	Brief Definition
GitHub	GitHub is an online platform that is constructed on the foundation of Git. It provides hosting for Git repositories, as well as features for collaboration and tools for managing projects. It is extensively utilized for both open-source and private development projects.
GitLab	GitLab is an online platform for managing Git repositories that offers features such as source code management, continuous integration/continuous deployment (CI/CD), and project planning. The software comprises functionalities for code evaluation, problem monitoring, and release administration.
Bitbucket	Bitbucket is a service provided by Atlassian that hosts Git repositories. It offers source code management, collaboration functionalities, and seamless connection with other Atlassian products such as Jira for tracking issues.
AWS CodeCommit	It is a comprehensive and supervised platform that provides hosting for GIT repositories, ensuring source control and security.
Jira	Jira, created by Atlassian, is a widely used software for managing projects and monitoring issues. It is extensively utilized for agile project management, enabling teams to quickly plan, track, and manage their work.
Trello	Trello is a graphical application for managing projects that use boards, lists, and cards to assist teams in arranging and ranking their tasks. It is renowned for its straightforwardness and adaptability in work management.
Asana	Asana is a platform for collaborative work management, enabling teams to efficiently organize and monitor their tasks. The software offers functionalities for project planning, task administration, and collaborative work.

Table 15: *TD Tools for Repository and Project Management*

4.13. Evaluation of TD Tools in Repository and Project Management Category

According to the literature reviews about the tools given at Table 15 and approximately 100 reviews written by users; main features, strong and weak points of these tools are given in Table 16.

<table>
<tr><th>Tool Name</th><th>Brief Definition</th></tr>
<tr><td>GitHub</td><td>Pros:
• Best for collaborative development
• It allows developers to host, review, and manage code and track and resolve issues
• It provides pull requests that facilitate the process of reviewing and merging modifications. Users utilize forks to create a duplicate of a repository in order to suggest modifications to the original version, and have the ability to include other GitHub users in the repository.
• GitHub pages to host a website for the project
• Built-in security features to secure code
• Automate tasks like testing, building, and deploying code
• The "gists" feature allows users to exchange concise fragments of code or text with others.
• Project boards facilitate effortless organization and prioritization of tasks. Additionally, users have the capability to contribute documentation and material to the project using wikis.
Cons:
• Built on top of Git, and users must know Git commands
• Issues with privacy and security in the past
OS: Docker over Windows, macOS, Linux, and Azure
Pricing: From $3.67/user/month (billed annually)
Trial: 30-day free trial + Free plan available
Official Sites: https://github.com/</td></tr>
</table>

<table>
<tr><td>GitLab</td><td>
<u>Pros:</u>

Best reporting features
It is an open-source code repository platform
Supports DevOps and CI/CD pipelines
Provides in-depth reports
Code controls reduce accidental changes to the code base
GitLab's Code Quality functionality facilitated the maintenance of clean, consistent, and manageable code for users. The tool performs code analysis after any modifications, including those made in merge requests, and provides an assessment of how the code quality has been affected prior to submitting the changes to the main branch.

<u>Cons:</u>

Limited integrations
Complex UI

<u>Pricing</u>:

Free version available
Premium version $29 per user
Ultimate version – no price info

<u>Official Sites</u>: https://about.gitlab.com/
</td></tr>
<tr><td>Bitbucket</td><td>
<u>Pros:</u>

Best for version control
Bitbucket is a robust Git solution that provides a platform that is easy to use. Additionally, it offers seamless connection with other Atlassian tools and robust features for team communication and software development.
This software is compatible with both Git and Mercurial version management systems. Users can utilize branching and merging capabilities to effectively handle codebase modifications and uphold superior code quality.
The free limitless private repositories facilitate collaborative work without requiring costly enterprise-level solutions.

</td></tr>
</table>

<table>
<tr><td></td><td>

Integrated CI/CD solution that allows you to build, test, and deploy your applications automatically
Unlimited pull requests reviewers
Its access control, allowing administrators to manage team member permissions on a per-repository basis.
With Git-based version control, users can get a fault-resistant distributed architecture that helps reduce single points of failure and minimize downtime in the event of a disaster.
Integrations are available natively for Jira, Trello, Slack, Amazon CodeGuru, Bugsnag, Buddybuild, CircleCI, CloudBees, and GuardRails.

<u>Cons:</u>

Limited storage space for large files
Does not support pull requests across forks in the free version

<u>Pricing</u>:

Standard version $3 per user $15 monthly total
Premium version $6 per user $30 monthly total

<u>Trial</u>: Free plan available
<u>Official Sites</u>: https://www.atlassian.com/software/bitbucket
</td></tr>
<tr><td>AWS CodeCommit</td><td>
<u>Pros:</u>

It utilizes many AWS services that customers can employ for code evaluations.
Access to the code can be managed by users, time, and location through the utilization of AWS Identity and Access Management (IAM) and Key Management Service (KMS).
Users can create repositories using their preferred manner, whether it is through AWS SDKs, CLI, or the Management Console. Users have the ability to closely monitor the repositories in real-time using CloudTrail and CloudWatch.
Easy to setup on AWS
Native integrations for AWS products and services
Robust user access control

</td></tr>
</table>

<table>
<tr><td></td><td>Cons:
Limited non-AWS integrations
Git functionality not as refined as alternatives like GitHub
Pricing:
5 active users per month for free, $1.00 per additional active user per month.
Free plan available
Official Sites: https://aws.amazon.com/tr/codecommit/</td></tr>
<tr><td>Jira</td><td>Pros:
Jira is a project management solution designed to facilitate collaborative planning, tracking, and management of work for teams.
Jira is highly useful for tracking issues, since it allows users to effortlessly generate issues, allocate tasks to team members, and prioritize work according to its severity.
It enables the monitoring of progress in real-time through the use of customizable dashboards and reports.
The platform offers agile tools such as Kanban and Scrum boards for the purpose of visualizing progress.
The software has a specialized query language called "JQL" that allows users to sort and filter issues based on various parameters. Additionally, there is a drag-and-drop functionality available for constructing epics and sprints within the backlog.
Advanced search features
Comprehensive activity log
Issue templates available
Integrations include native options like Balsamiq, Zendesk, Zephyr, EazyBI, Salesforce Sales Cloud, Atlassian Confluence, and nFeed.
Cons:
Requires technical expertise to utilize advanced features fully
Some users find the interface cluttered or overwhelming</td></tr>
</table>

	<u>Pricing</u>: • Standard version $8.15 per user $81.50 monthly total • Premium version $16 per user $160 monthly total • Enterprise edition – annually billed <u>Trial</u>: Free plan available <u>Official Sites</u>: https://www.atlassian.com/software/jira
Trello	<u>Pros:</u> • Best for visual Kanban organization within small teams • It is famous for its utilization of Kanban-style boards, lists, and cards, which facilitate the organization of activities • Simple, intuitive Kanban system • Flexible and responsive to varied workflows • Good collaboration features • Integrations include Google Drive, Slack, Jira, GitHub, and Dropbox natively, and can integrate with more tools via Zapier. APIs include REST API, Webhooks API, and Power-Ups API <u>Cons:</u> • There are limited reporting tools compared to more sophisticated platforms • The dashboard can feel cluttered when projects become too complex <u>Pricing</u>: • Free version is available • Standard version $5 per user/month • Premium version $10 per user/month • Enterprise version – $17.50 per user/month <u>Official Sites</u>: https://trello.com/
Asana	<u>Pros:</u> • Asana is a widely used alternative to Jira. Designed to prioritize visual task management and facilitate team coordination, this tool is particularly well-suited for teams seeking efficient project structure, monitoring, and management

	• Highly-visual task management • A range of collaboration features • Free tier and affordable premium packages for small companies • Integrations include Slack, Google Workspace, Microsoft Office 365, Salesforce, and Adobe Creative Cloud. Plus, you can integrate with more tools via Zapier. APIs include REST API, Webhooks API, and Tasks API • The software offers Kanban boards and Gantt charts to facilitate project visualization. Cons: • Range of features can be overwhelming • Advanced features are quite complex to use Pricing: • Free version is available • Starter version $10.99 per user/month • Advanced version $24.99 per user/month • Enterprise versions – no price info Official Sites: https://asana.com/

Table 16: *Evaluation of TD Tools for Repository and Project Management*

CHAPTER 5

CONCLUSION

The exploration of technical debt and related management techniques has shown the crucial significance of tackling this widespread concern in software development. As emphasized in this term project, technical debt manifests in different ways, ranging from design concessions to postponed testing, each impacting the development process and the end result.

The assessment of various technologies specifically designed for managing technical debt has yielded significant information for professionals traversing the intricate terrain of software development. By conducting a meticulous examination of the advantages and disadvantages, we have elucidated the positive and negative aspects of each tool, providing a nuanced comprehension of their suitability in various scenarios. This project functions as a pragmatic manual for development teams and decision-makers, empowering them to make well-informed decisions that are customized to their specific project needs.

This study focuses on developing an understanding of technical debt and the tools used for managing it. Through a literature review and analysis of popular blogs, we explore 30 different tools that are commonly utilized across 6 distinct categories. After extensive review of numerous academic papers and approximately 500 evaluations on reputable websites, conducted by experts in the field, we have successfully determined the strengths, flaws, and significant characteristics of each tool under examination.

Based on practical and technological research, no tool has been identified as the definitive favorite, as none of the tools have proven to be the most successful in every element. The factors to consider are outlined as the crucial elements that decide the choice of tool, which varies between organizations and depending on the project's level of complexity and domain. For example, in small-scale projects it would not be appropriate to use applications such as "CAST" or "Micro Focus Fortify Static Code Analyzer", which have very comprehensive features. For this reason, when choosing an application, factors such as the scale of the company and the developed project and domain requirements should be taken into consideration.

In addition, based on the software project development life cycle, the most important issue that will affect the success of the project or cause serious costs later is architectural design. For this reason, it is very important to develop software in accordance with coding standards and SOLID principles during the software development process.

In conclusion, the findings presented here emphasize the necessity of a holistic approach to technical debt. It is not merely a code quality concern but a crucial factor influencing the overall success and sustainability of software projects. This paper aims to provide organizations and users with a clear understanding of the concept of technical debt and the ability to differentiate between various tools when managing the lifecycle of a software project.

The work completed for this report can be expanded upon to discuss additional applications and, given the wide range of applications, analyze in-depth the performance of appropriate tools for a given need in projects of varying sizes. Additionally, the tools under the various categories specified in this study can be thoroughly examined.

REFERENCES

[1] Verdecchia, R., Malavolta, I., & Lago, P. (2018, May). Architectural technical debt identification: The research landscape. In *Proceedings of the 2018 International Conference on Technical Debt* (pp. 11-20).

[2] Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., & Fontana, F. A. (2019). Technical debt prioritization: State of the art. A systematic literature review. *arXiv preprint arXiv:1904.12538*.

[3] Behutiye, W. N., Rodríguez, P., Oivo, M., & Tosun, A. (2017). Analyzing the concept of technical debt in the context of agile software development: A systematic literature review. *Information and Software Technology*, *82*, 139-158.

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

[5] Avgeriou, P. C., Taibi, D., Ampatzoglou, A., Fontana, F. A., Besker, T., Chatzigeorgiou, A., ... & Tsintzira, A. A. (2020). An overview and comparison of technical debt measurement tools. *Ieee software*, *38*(3), 61-71.

[6] Saraiva, D., Neto, J. G., Kulesza, U., Freitas, G., Reboucas, R., & Coelho, R. (2021). Technical Debt Tools: A Systematic Mapping Study. *ICEIS (2)*, 88-98.

[7] Stochel, M. G., Chołda, P., & Wawrowski, M. R. (2020, August). On coherence in technical debt research: Awareness of the risks stemming from the metaphorical origin and relevant remediation strategies. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 367-375). IEEE.

[8] BenIdris, M. (2020). Investigate, identify and estimate the technical debt: a systematic mapping study. *Available at SSRN 3606172*.

[9] Wiese, M., Riebisch, M., & Schwarze, J. (2021, May). Preventing Technical Debt by Technical Debt Aware Project Management. In *2021 IEEE/ACM International Conference on Technical Debt (TechDebt)* (pp. 84-93). IEEE.

[10] Ernst, N. A., Bellomo, S., Ozkaya, I., Nord, R. L., & Gorton, I. (2015, August). Measure it? manage it? ignore it? software practitioners and technical debt. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (pp. 50-60).

[11] Yli-Huumo, J., Maglyas, A., & Smolander, K. (2016). How do software development teams manage technical debt?–An empirical study. *Journal of Systems and Software*, *120*, 195-218.

[12] Ampatzoglou, A., Mittas, N., Tsintzira, A. A., Ampatzoglou, A., Arvanitou, E. M., Chatzigeorgiou, A., ... & Angelis, L. (2020). Exploring the relation between technical debt principal and interest: An empirical approach. *Information and Software Technology*, *128*, 106391.

[13] Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., & Fontana, F. A. (2021). A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools. Journal of Systems and Software, 171, 110827.

[14] Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., & Fontana, F. A. (2021). A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools. *Journal of Systems and Software*, *171*, 110827.

[15] Melo, A., Fagundes, R., Lenarduzzi, V., & Santos, W. B. (2022). Identification and measurement of Requirements Technical Debt in software development: A systematic literature review. *Journal of Systems and Software*, *194*, 111483.

[16] Soliman, M., Avgeriou, P., & Li, Y. (2021). Architectural design decisions that incur technical debt—An industrial case study. *Information and Software Technology*, *139*, 106669.

[17] Cunningham, W. (1992). The WyCash portfolio management system. *ACM Sigplan Oops Messenger*, *4*(2), 29-30.

[18] Besker, T., Martini, A., & Bosch, J. (2018). Managing architectural technical debt: A unified model and systematic literature review. *Journal of Systems and Software*, *135*, 1-16.

[19] Martini, A. (2018, May). Anacondebt: a tool to assess and track technical debt. In *Proceedings of the 2018 International Conference on Technical Debt* (pp. 55-56).

[20] von Zitzewitz, A. (2019, May). Mitigating technical and architectural debt with sonargraph. In *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)* (pp. 66-67). IEEE.

[21] Tornhill, A. (2018, May). Prioritize technical debt in large-scale systems using CodeScene. In *Proceedings of the 2018 International Conference on Technical Debt* (pp. 59-60).

[22] Verdecchia, R., Kruchten, P., Lago, P., & Malavolta, I. (2021). Building and evaluating a theory of architectural technical debt in software-intensive systems. *Journal of Systems and Software*, *176*, 110925.

[23] Pavlič, L., & Hliš, T. (2019). The Technical Debt Management Tools Comparison . In *Eighth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications* (p. 10).

[24] Rios, N., Spinola, R. O., de Mendonça Neto, M. G., & Seaman, C. (2018, August). A study of factors that lead development teams to incur technical debt in software projects. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 429-436). IEEE.

[25] Kruchten, P., Nord, R., & Ozkaya, I. (2019). Managing Technical Debt: Reducing Friction in Software Development. Software Engineering Institute.

[27] Atlassian Continuous Integration Tools: Top 7 Comparison, https://www.atlassian.com/continuous-delivery/continuous-integration/tools, Date of Access 29.12.2023.

[28] Comparitech 10 Best DAST Tools, https://www.comparitech.com/net-admin/dast-tools/, Date of Access 29.12.2023.

[29] Gartner Best Application Security Testing Reviews 2023: Gartner Peer Insights, https://www.gartner.com/reviews/market/application-security-testing, Date of Access 29.12.2023.

[30] PeerSpot Top Rated Dynamic Application Security Testing (DAST) Vendors, https://www.peerspot.com/categories/dynamic-application-security-testing-dast, Date of Access 29.12.2023.

[31] The CTO Club 20 Best Code Review Tools for Developers [2023 Guide], https://thectoclub.com/tools/best-code-review-tools/, Date of Access 29.12.2023.

[32] Katalon Top 15 Automation Testing Tools 2024, https://katalon.com/resources-center/blog/automation-testing-tools, Date of Access 29.12.2023.

[33] Sonar Clean Code Tools for Writing Clear, Readable & Understandable Secure Quality Code, https://www.sonarsource.com/, Date of Access 29.12.2023.

[34] Jenkins, https://www.jenkins.io/, Date of Access 29.12.2023.

[35] Amazon Web Services AWS CodePipeline, https://aws.amazon.com/tr/codepipeline/, Date of Access 29.12.2023.

[36] LambdaTest Next-Generation Mobile Apps and Cross Browser Testing Cloud, https://www.lambdatest.com/, Date of Access 29.12.2023.

[37] Cypress JavaScript Component Testing and E2E Testing, https://www.cypress.io/, Date of Access 29.12.2023.

[38] Ndepend Improve your .NET code quality with NDepend, https://www.ndepend.com/, Date of Access 29.12.2023.

[39] Structure101-Software Architecture Development, https://structure101.com/, Date of Access 29.12.2023.

[40] JArchitect Java Static Analysis and Code Quality Tool, https://www.jarchitect.com/, Date of Access 29.12.2023.

[41] Intruder, https://www.intruder.io/, Date of Access 29.12.2023.

[42] SOOS DAST Product, https://soos.io/, Date of Access 29.12.2023.

[43] Invicti Web Application Security, https://www.invicti.com/, Date of Access 29.12.2023.

[44] Checkmarx Application Security Testing Company Software, https://checkmarx.com/, Date of Access 29.12.2023.

[45] Synopsys Coverity Static Analysis Software, https://www.synopsys.com/, Date of Access 29.12.2023.

[46] CAST Highlight, https://www.castsoftware.com/, Date of Access 29.12.2023.