

Department of Computer Science – IT

DCIT 415: Advanced Software Engineering 1st Semester

ID Number: 10817756

Systematic Literature Review On Self-Admitted Technical Debt

# Self-Admitted Technical Debt

## Systematic Literature Review on Self-Admitted Technical Debt

### Abstract:

Technical debt is a well-known concept in software engineering that describes the consequences of choosing an easier or faster approach to software development that results in long-term costs or consequences. Self-admitted technical debt is a type of technical debt that is explicitly acknowledged by software developers within the code, comments, or documentation.

The systematic literature review conducted on self-admitted technical debt aimed to identify research gaps and opportunities by analyzing 20 selected articles that covered various aspects of self-admitted technical debt, such as its perception, nature, measurement, impact, and detection.

The review revealed that self-admitted technical debt is prevalent in software development and can have a significant impact on the quality and maintainability of software systems. However, more research is needed to better understand the factors that influence the occurrence and management of self-admitted technical debt, and to develop effective tools and techniques for its detection and mitigation. (Alves, C. A., & Brito, I. A. 2020 p. 167)

In conclusion, self-admitted technical debt is a critical issue in software development that needs to be addressed effectively. The literature review highlighted the need for more research in this area, particularly in understanding the causes and effects of self-admitted technical debt and developing tools and techniques to mitigate its impact.

Future work should focus on exploring the use of machine learning and data analytics to identify self-admitted technical debt, as well as developing strategies for managing technical debt in different software development environments. Additionally, further studies are needed to investigate the relationship between self-admitted technical debt and software quality, as well as the cost of managing technical debt in the long term.

### Introduction:

Technical debt is a critical challenge faced by software development teams. It refers to the extra cost that development teams incur when they choose to take shortcuts or make compromises during the development process. These shortcuts or compromises may lead to the accumulation of technical debt, which can have negative impacts on the software systems' quality, reliability, and maintainability.

Self-admitted technical debt is a type of technical debt that is identified by developers within the code, comments, or documentation. It is a critical aspect of software development that can have significant implications on the project's success. The management of self-admitted technical debt is crucial to the project's success, and identifying and managing it requires specific tools and techniques.

This paper presents a systematic literature review on self-admitted technical debt, aiming to provide a comprehensive analysis of the current state-of-the-art, identify research gaps, and suggest future research directions. The review analyzed twenty articles covering various aspects of self-admitted technical debt, including its perception, nature, measurement, impact, and detection.

The analysis revealed that self-admitted technical debt is prevalent in software development, and it can have significant impacts on the quality and maintainability of software systems. The review identified several factors that contribute to the occurrence and management of self-admitted technical debt, including team culture, development practices, and project scope. The review also highlighted the need for further research to better understand the factors that influence the occurrence and management of self-admitted technical debt, and to develop effective tools and techniques for its detection and mitigation.

In conclusion, self-admitted technical debt is a critical challenge faced by software development teams. The review highlighted the prevalence and impact of self-admitted technical debt, the factors that influence its occurrence and management, and the need for further research to better understand and manage this type of technical debt. The review provides a useful resource for researchers and practitioners interested in self-admitted technical debt, highlighting research gaps, and suggesting future research directions.

### Selection Criteria:

The selection criteria used in this systematic literature review (SLR) are essential to ensure that only relevant and high-quality articles are included in the analysis.

* The first criterion, that the article should focus on self-admitted technical debt, is essential for the SLR to stay focused on the specific type of technical debt of interest. By limiting the search to articles that explicitly discuss self-admitted technical debt, the review can provide a comprehensive and in-depth analysis of this type of technical debt. Articles that focus on other types of technical debt, such as architectural debt, design debt, or testing debt, may not provide sufficient information on self-admitted technical debt, leading to potential gaps in the analysis. Therefore, by focusing on self-admitted technical debt, the SLR can ensure that the review provides a thorough understanding of the specific topic of interest, which can lead to more precise conclusions and recommendations for future research.
* The second criterion of the SLR, that the article should be written in English, is necessary to ensure that language barriers do not exclude relevant articles from the analysis. Although scientific research is conducted worldwide, English is the primary language of communication in most scientific fields. Most high-quality journals and conferences require authors to submit their papers in English, and many researchers are also accustomed to publishing their work in English-language outlets. Therefore, by limiting the search to articles written in English, the SLR ensures that it can analyze a more comprehensive set of articles, including those from different countries and research institutions. This criterion can also help ensure that the findings and conclusions of the SLR are accessible to a broader audience, including researchers, software practitioners, and policymakers, who may rely on English-language publications as a primary source of information.
* The third criterion is crucial to ensure that the articles included in the analysis meet the quality standards of the scientific community. Peer-reviewed conferences or journals provide a thorough review process in which experts in the field evaluate the research quality and its contribution to the field. This process helps to ensure that the articles meet the quality standards for publication in a scientific journal or conference.

Peer-review is a process in which an article or research paper is evaluated by experts in the same field to ensure its quality, accuracy, and relevance. Peer-review is a crucial aspect of the scientific research process as it helps to ensure that the research published is reliable, valid, and contributes to the knowledge base of the field.

Peer-review involves submitting an article or research paper to a journal or conference for evaluation by other experts in the field. These experts, known as peers, review the paper and provide feedback on its quality, accuracy, and relevance. The feedback can include suggestions for revisions or improvements to the paper.

Peer-reviewed articles are typically considered to be of higher quality than articles that are not peer-reviewed. This is because peer-review ensures that the article has undergone a rigorous evaluation by experts in the field, who have assessed its quality, accuracy, and relevance. Peer-reviewed articles are also more likely to be published in reputable journals or conferences, which further adds to their credibility.

In contrast, articles that are not peer-reviewed may not undergo the same level of scrutiny and may not have been evaluated by experts in the field. This can result in lower quality research or studies that do not contribute significantly to the field of interest. Non-peer-reviewed articles may also lack the credibility and validity that comes with peer-review, which can make them less reliable sources of information.

In summary, peer-review is an essential aspect of the scientific research process, as it helps to ensure the quality, accuracy, and relevance of research published in academic journals or conferences. Articles that are not peer-reviewed may be of lower quality and may lack the credibility and validity that comes with peer-review. Therefore, it is important to critically evaluate the sources of information to ensure that they are dependable and of high quality.

By including only peer-reviewed articles in the analysis, the systematic literature review can ensure that the studies selected are of high quality and have been validated by experts in the field. This helps to strengthen the reliability and validity of the findings and conclusions drawn from the analysis.

* The fourth criterion, that the article should be published between 2010 and 2021, is essential to ensure that the review covers the most recent research in the field. In the fast-changing field of software engineering, it is important to keep up-to-date with the latest developments, and by including articles published within the past 11 years, the review can provide an up-to-date and comprehensive analysis of the current state-of-the-art.

This criterion also allows the review to identify recent trends in the field of self-admitted technical debt, such as the emergence of new tools and techniques for detecting and managing technical debt, as well as changes in the perception and understanding of the phenomenon over time.

Additionally, limiting the search to the past 11 years helps to ensure that the articles included in the review are still relevant and applicable to current software development practices. Older articles may be less useful, as they may not reflect current practices, methodologies, or technologies. Therefore, by focusing on articles published within the past 11 years, the SLR can provide a more relevant and accurate analysis of the current state-of-the-art in self-admitted technical debt research.

Overall the selection criteria used in this systematic literature review (SLR) ensure that the articles included in the analysis are relevant, high-quality, and up-to-date. By focusing only on articles that specifically address self-admitted technical debt, the SLR provides a more in-depth understanding of this specific type of technical debt. The requirement that articles be written in English helps to ensure that language barriers do not exclude relevant articles from the analysis, as English is the primary language of communication in scientific research. Additionally, the requirement that articles be published in peer-reviewed conferences or journals helps to ensure the quality of the articles included in the analysis, as these articles have undergone a rigorous review process by experts in the field. Finally, by limiting the search to articles published between 2010 and 2021, the SLR provides a comprehensive analysis of the most recent research in the field, identifying recent trends and research gaps. Overall, these selection criteria ensure that the SLR provides a robust and up-to-date analysis of the current state-of-the-art in self-admitted technical debt research.

### Selection Databases:

The databases mentioned in the systematic literature review (SLR) are widely recognized as reputable sources of scientific literature in the field of computer science and software engineering.

* The IEEE Xplore Digital Library is a leading database for technical literature in the field of electrical engineering, computer science, and related disciplines. It offers access to a vast collection of IEEE journals, conference proceedings, and standards. The database is curated by the Institute of Electrical and Electronics Engineers (IEEE), a professional organization dedicated to advancing technology for the benefit of humanity.

IEEE Xplore is a valuable resource for researchers and practitioners who are seeking high-quality technical literature. The database covers a wide range of topics, including artificial intelligence, data science, cybersecurity, communication networks, and more. It also provides advanced search capabilities, allowing users to refine their search by various criteria such as publication type, publication year, author, and more.

Overall, the IEEE Xplore Digital Library is a trusted source of scholarly literature in the field of electrical engineering, computer science, and related disciplines, and it is widely used by researchers, academics, and industry professionals worldwide.

* The ACM Digital Library is a comprehensive database that contains various publications in the field of computing and information technology, including journals, conference proceedings, magazines, and newsletters. ACM, the Association for Computing Machinery, is the world's largest scientific and educational computing society, which provides a wide range of resources to support the field of computing.

The ACM Digital Library is considered one of the most prestigious and respected sources for scientific research in the field of computing, covering a wide range of topics such as computer architecture, human-computer interaction, programming languages, artificial intelligence, data science, and more. It is widely used by researchers, scholars, and practitioners in academia and industry worldwide.

The ACM Digital Library is known for its high-quality peer-reviewed content and provides access to more than 1 million articles, books, and conference proceedings from ACM and other publishers. It also includes various tools and services such as advanced search options, citation management tools, and personalized recommendations, which help users to discover relevant and up-to-date research in their fields of interest.

* Scopus is a bibliographic database of peer-reviewed literature that provides comprehensive coverage of scientific, technical, medical, and social sciences fields. It is one of the largest abstract and citation databases, covering more than 70 million documents from over 5,000 publishers worldwide. Scopus provides access to scholarly articles, conference proceedings, book chapters, and other types of research publications. It also offers advanced search and analysis tools, such as citation tracking, author profiling, and co-citation analysis, which can help researchers identify key trends and influencers in their field of study. Scopus is widely used by researchers, librarians, and institutions to support research and scholarly communication. It is maintained by Elsevier, a global information analytics company specializing in science, health, and technology content.
* Web of Science is a database that provides access to a vast collection of peer-reviewed journals, conference proceedings, and other scholarly resources in various disciplines, including science, social sciences, arts, and humanities. It is one of the oldest and most well-known sources of scientific literature and is widely used by researchers and scholars in academia. The database includes the Science Citation Index, the Social Sciences Citation Index, and the Arts & Humanities Citation Index, which cover different fields of study. Web of Science provides citation indexing, citation analysis, and other bibliometric analysis tools, making it a useful resource for researchers to analyze the impact of their work and identify emerging trends in their field. Overall, Web of Science is a valuable resource for conducting research and staying up-to-date on the latest developments in various academic disciplines.

The process of searching multiple databases is a common practice in systematic literature reviews to ensure comprehensive coverage of relevant publications. In this case, the author of the SLR searched four prominent databases, including IEEE Xplore, ACM Digital Library, Scopus, and Web of Science. These databases are known to provide access to a vast collection of peer-reviewed publications in various fields, including computer science, software engineering, and related areas.

By searching these databases, the author could access a broad range of relevant publications, including peer-reviewed conference papers and journal articles, to cover the research on self-admitted technical debt thoroughly. This approach reduces the possibility of missing relevant publications, as different databases may index different publications. Additionally, using multiple databases improves the quality of the search results, as the author can cross-check the search results across multiple sources to ensure the relevance and accuracy of the included publications.

### Search Process:

The search terms used in an SLR play a crucial role in identifying relevant articles that meet the inclusion criteria. In this case, the following search terms were used:

* "self-admitted technical debt"
* "technical debt"
* "code smells"
* "refactoring"
* "software quality"

These terms were chosen based on their relevance to the topic of self-admitted technical debt. "Self-admitted technical debt" is a specific type of technical debt that is recognized by software developers within the code, comments, or documentation. "Technical debt" is a broader term that refers to any compromises or shortcuts taken during the development process that can lead to reduced software quality. "Code smells" are indicators of poor code quality that suggest the presence of technical debt. "Refactoring" is a process that aims to improve software quality by eliminating technical debt. "Software quality" is a general term that encompasses various aspects of software development, including technical debt.

I used multiple search terms to capture different aspects of self-admitted technical debt and related topics, which could have been missed if only one search term was used. For example, the search term "code smells" is a common term used to refer to code anomalies that indicate the presence of technical debt in the code. The term "refactoring" is also relevant as it is a common technique used to address technical debt in software systems. By using these search terms, the author was able to identify a more comprehensive set of articles related to self-admitted technical debt, leading to a more comprehensive analysis of the current state-of-the-art in the field.

### Data Extraction Process:

The information extracted for each selected article provides a comprehensive understanding of the study and its outcomes. Here is an elaboration of the information extracted:

* Title: The title of the article is the name given to it by the authors and typically provides a brief summary of the main topic or focus of the research. The title of an article is a concise statement that reflects the main topic or focus of the research. It is typically provided by the authors and appears at the beginning of the article. The title is usually the first thing that readers see, and it helps them determine whether the article is relevant to their interests.

A good title should be clear, concise, and informative. It should accurately reflect the content of the article and attract the attention of potential readers. A well-crafted title can increase the visibility and impact of an article, as it can make it easier for readers to find and cite the research.

When writing a title, authors should consider the following questions: What is the main focus or topic of the research? What are the key findings or conclusions? Who is the target audience? What is the significance of the research?

Authors should also ensure that the title adheres to the guidelines and requirements of the publication or journal to which they are submitting. Many journals have specific formatting and length requirements for titles, and authors should carefully review these guidelines before submitting their articles.

* Authors: The authors' names are a crucial piece of information that allows for the identification of the researchers responsible for conducting the study. The authors' names typically appear at the beginning of the article, and they may be listed in order of their contributions to the research.

Knowing the authors' names is important for several reasons. First, it allows readers to give credit to the researchers for their work. Second, it helps readers evaluate the expertise and authority of the authors, as their credentials and affiliations may be listed alongside their names. Third, it allows readers to contact the authors if they have questions or wish to collaborate on future research.

In addition to the authors' names, other important information about the authors may be included in the article. This may include their academic degrees, professional titles, institutional affiliations, and contact information. These details can help readers assess the credibility and reliability of the authors and their research.

When listing the authors' names, it is important to follow the conventions of the publication or journal. Some publications may require authors to use their full names, while others may allow the use of initials or nicknames. Additionally, authors may be required to indicate their specific contributions to the research or to provide a brief bio or statement about themselves.

* Publication venue: The name of the conference or journal where an article is published is a crucial piece of information that provides important insights into the article's intended audience, research community, and level of significance. This information is typically listed near the beginning of the conference or journal where an article is published can be an indication of the level of rigor and quality of the research. Prestigious and well-established journals are often more selective in their acceptance of articles and have a rigorous peer-review process, which can indicate that the research has undergone thorough scrutiny and is of high quality. In contrast, lower-tier journals or conferences may have less rigorous acceptance criteria and may not attract the same level of scrutiny or attention from the academic community.

In addition to the level of significance, the name of the conference or journal can also provide insights into the article's intended audience and research community. For example, articles published in highly specialized and niche journals or conferences may be targeted towards a specific community of researchers or practitioners, while articles published in more general journals may be intended for a broader audience.

Authors may also choose to publish in certain conferences or journals based on their reputation and visibility within their field. This can be important for building credibility and establishing a reputation as a researcher within a particular research community.

* Publication year: The year in which an article was published provides valuable information about its relevance and recency. In general, articles that have been published more recently are likely to be more relevant to current events and trends, and may include the latest research or insights in a particular field.

For example, if you are researching a topic related to technology, an article published in 2022 is likely to be more up-to-date and relevant than an article published in 2010. This is because technology changes rapidly, and newer articles are likely to reflect the most recent advancements and trends in the field.

Similarly, if you are researching a topic related to public policy or politics, articles published in the most recent year may be particularly relevant, as they are likely to reflect the latest developments and debates on the issue.

On the other hand, if you are researching a historical event or a long-term trend, older articles may be more relevant, as they can provide important context and perspective on the topic.

* Research questions or objectives: Understanding the primary research question or objective of a study is crucial for properly evaluating the significance and relevance of the study's findings. The research question or objective is the main problem or issue that the study aims to investigate or address. It serves as the guiding principle for the research and is typically stated in the introduction or methods section of the study.

Knowing the research question or objective helps readers to understand what the researchers were trying to accomplish with their study, and what specific questions or hypotheses they were testing. This information is important because it provides context for the study's findings, and helps readers to determine whether the study is relevant to their own research interests or questions.

For example, if you are interested in the effects of a particular drug on a specific health condition, knowing the primary research question or objective of a study on that drug will help you to determine whether the study's findings are applicable to your interests. If the study's research question was focused on a different health condition, for example, its findings may not be as relevant to your interests.

In addition, understanding the research question or objective of a study can help readers to evaluate the quality and rigor of the study. If the research question is well-defined and focused, and the study's methods are designed to address that question in a systematic and rigorous way, the study is more likely to produce reliable and valid results.

* Research methods: Describing the research methods used in a study is important for understanding how the study was conducted and how the data were collected and analyzed. The research methods section of a study typically provides a detailed description of the study design, data collection procedures, and data analysis techniques.

The data collection methods used in a study can vary widely depending on the research question and the nature of the data being collected. Some common data collection methods include surveys, interviews, experiments, observational studies, and case studies. The research methods section should provide a clear description of the data collection methods used in the study, including any specific techniques or instruments used to collect the data.

In addition to data collection, the research methods section should also describe the data analysis techniques used in the study. This may include statistical methods, qualitative analysis techniques, or a combination of both. The analysis methods used in a study should be appropriate for the type of data being analyzed and should be clearly explained in the research methods section.

Understanding the research methods used in a study is important for evaluating the quality and reliability of the study's findings. The methods used to collect and analyze data can have a significant impact on the validity and generalizability of the study's results. By providing a detailed description of the research methods used, readers can assess whether the study's methods were appropriate for the research question and whether the results are dependable and meaningful.

* Key findings: Describing the primary outcomes or key findings of a study is important for understanding the main results and conclusions of the study. The primary outcomes or key findings are the main results or conclusions that the study aimed to investigate or address.

The primary outcomes or key findings section of a study typically provides a summary of the main results, including any statistical significance, effect sizes, or confidence intervals associated with the findings. This section should be clear and concise, and should avoid overly technical language or jargon.

Understanding the primary outcomes or key findings of a study is important for evaluating the relevance and significance of the study's results. It can also help readers to determine whether the study's results are applicable to their own research interests or questions.

For example, if you are interested in the effects of a particular intervention on a specific health outcome, knowing the primary outcomes or key findings of a study on that intervention will help you to determine whether the intervention was effective in improving the health outcome of interest. If the study's primary outcomes or key findings indicate that the intervention was not effective, for example, you may need to consider other interventions or approaches to achieve your research goals.

In addition, understanding the primary outcomes or key findings of a study can help readers to evaluate the quality and rigor of the study. If the study's primary outcomes or key findings are clear and well-supported by the data, and the study's methods are designed to address the research question in a systematic and rigorous way, the study is more likely to produce reliable and valid results.

* Limitations: Outlining the limitations of a study and potential sources of bias or areas where the research could have been improved is important for understanding the strengths and weaknesses of the study. Every study has limitations, and it is important to acknowledge and address these limitations to evaluate the quality and reliability of the study's findings.

The limitations section of a study typically provides a description of any potential sources of bias or areas where the research could have been improved. This may include limitations in the study design, data collection methods, data analysis techniques, or sample size. The limitations section should be clear and concise, and should provide a balanced assessment of the study's strengths and weaknesses.

Understanding the limitations of a study is important for interpreting the study's findings and for evaluating the relevance and applicability of the results. For example, if the study had a small sample size or limited geographic scope, it may not be possible to generalize the results to a larger population or to other settings.

In addition, understanding the limitations of a study can help readers to evaluate the quality and rigor of the study. By acknowledging and addressing potential sources of bias or limitations in the study design, the researchers can demonstrate their awareness of these issues and their commitment to producing reliable and valid results.

* Future research directions: Suggesting potential areas for future research based on the findings of the study and any limitations or gaps identified is important for advancing the field and building on the existing knowledge base. The discussion section of a study typically provides an opportunity for the authors to highlight the implications of their findings for future research.

In this section, the authors may suggest potential avenues for further investigation, based on the limitations or gaps identified in the study. For example, if the study found that a particular intervention was not effective, the authors may suggest that future research investigate alternative interventions or examine the factors that contributed to the lack of effectiveness.

Alternatively, the authors may suggest areas for future research based on the potential implications of their findings. For example, if the study found that a particular intervention was effective, the authors may suggest that future research examine the long-term effects of the intervention, or investigate the factors that may influence the effectiveness of the intervention in different settings or populations.

Identifying potential areas for future research is important for advancing the field and building on the existing knowledge base. By highlighting the limitations or gaps in their own study, and suggesting potential avenues for further investigation, researchers can help to guide future research and contribute to the development of new interventions or approaches.

### Research Questions:

This systematic literature review aims to address five research questions related to self-admitted technical debt (SATD). Each research question focuses on a specific aspect of SATD and is designed to provide a comprehensive understanding of this phenomenon.

* The first research question of the systematic literature review aims to identify the characteristics of self-admitted technical debt (SATD) in software development projects. SATD refers to code implementations that developers themselves consider to be suboptimal or in need of improvement. These implementations are often the result of trade-offs made during software development, such as prioritizing speed of delivery over code quality, or working with legacy code that may be difficult to refactor.

By examining the characteristics of SATD, researchers can gain insights into the types of issues that lead to the accumulation of debt in software development projects. These insights can include the types of code issues that are most commonly associated with SATD, the frequency with which SATD occurs in different types of software projects, and the impact that SATD can have on software quality and maintenance.

Some of the characteristics of SATD that may be identified through research include the types of code smells or anti-patterns that are associated with SATD, such as duplicated code, long methods, or complex conditionals. Additionally, researchers may examine the factors that contribute to the accumulation of SATD, such as project complexity, tight deadlines, or a lack of resources for code review and refactoring.

By understanding the characteristics of SATD, researchers can help software development teams to better identify and manage technical debt, which can lead to improved software quality and reduced maintenance costs. For example, by identifying the most common types of SATD, developers can prioritize code reviews and refactoring efforts to address these issues. Similarly, by understanding the factors that contribute to the accumulation of SATD, organizations can take steps to address these underlying issues, such as providing more resources for code review or prioritizing code quality over speed of delivery.

* The second research question of the systematic literature review focuses on the factors that influence the occurrence and management of self-admitted technical debt (SATD) in software development projects. This question seeks to understand the underlying causes of SATD and the factors that make it difficult to address or manage. By identifying these factors, researchers can help developers and organizations to better manage and reduce the accumulation of technical debt.

Some of the factors that may influence the occurrence of SATD include project complexity, tight deadlines, lack of resources for code review and refactoring, and the pressure to deliver new features quickly. Developers may also be more likely to incur technical debt when they lack knowledge or experience with certain technologies or when working with legacy code.

1. Project Complexity: Projects with complex requirements or architectures may be more prone to technical debt. This is because developers may need to prioritize meeting functional requirements over code quality to deliver a working product.
2. Tight Deadlines: Projects with tight deadlines may also be more prone to technical debt. When developers are under pressure to deliver features quickly, they may resort to quick-and-dirty solutions instead of taking the time to write high-quality, maintainable code.
3. Lack of Resources for Code Review and Refactoring: Lack of resources, such as time or personnel, for code review and refactoring may also contribute to the accumulation of technical debt. When there are no resources to devote to these tasks, developers may be forced to cut corners to meet deadlines.
4. Pressure to Deliver New Features Quickly: The pressure to deliver new features quickly may also contribute to technical debt. When stakeholders are focused on delivering new features, developers may prioritize adding new functionality over maintaining code quality.
5. Lack of Knowledge or Experience with Certain Technologies: Developers who lack knowledge or experience with certain technologies may be more likely to incur technical debt. In these cases, developers may not know how to write optimal code or be aware of best practices that could help prevent technical debt.
6. Working with Legacy Code: Working with legacy code can also increase the likelihood of incurring technical debt. Developers may need to make changes to code that was not originally designed with modern coding practices in mind. This can make it challenging to write high-quality, maintainable code without incurring technical debt.

In addition to the factors that influence the occurrence of SATD, the second research question also focuses on the factors that make it difficult to manage technical debt once it has been incurred. For example, organizations may struggle to allocate resources for code review and refactoring, or may prioritize new feature development over addressing technical debt. Developers may also face challenges in identifying and addressing SATD due to a lack of documentation or a lack of understanding of the code base.

By understanding these factors, researchers can help developers and organizations to develop strategies for managing technical debt more effectively. For example, organizations may need to prioritize code quality over speed of delivery or allocate more resources for code review and refactoring.

* The third research question pertains to identifying the most effective tools and techniques that can be used to detect and manage Self-Admitted Technical Debt (SATD) in software projects. This question is important because SATD can lead to software quality issues and increase the maintenance burden on software developers.

To address this question, researchers can conduct a systematic review of existing literature and identify the most commonly used approaches for detecting and managing SATD. Some of the tools and techniques that can be evaluated include static code analysis, code smell detection, technical debt index, code churn analysis, and machine learning-based techniques.

Static code analysis is a technique used to analyze source code without executing it. It can help developers to identify potential issues related to coding standards, maintainability, and performance. Static code analysis tools can analyze source code and provide developers with feedback on code quality issues, such as potential bugs, coding style violations, and security vulnerabilities.

One important aspect of static code analysis is code smell detection. Code smells are common design issues that can lead to technical debt and other code quality problems. Code smells are usually not bugs, but rather design flaws or deviations from good coding practices that can make code harder to read, maintain, and scale. Code smells can be identified using a variety of techniques, including automated code analysis tools, manual code reviews, and code refactoring.

Some common code smells that can lead to technical debt include:

1. Duplication: When code is duplicated across different parts of a program, it can be harder to maintain and update. This can lead to inconsistencies in the codebase, as changes made to one part of the code may not be reflected in other parts.
2. Complexity: Code that is overly complex can be harder to read and understand. This can lead to errors and make it harder to maintain and update the codebase over time.
3. Poor naming conventions: Code that uses poor naming conventions can be harder to read and understand. This can make it harder for other developers to work with the code and increase the likelihood of errors.

By identifying code smells and other code quality issues through static code analysis, developers can take steps to address these issues and reduce the risk of technical debt. This may involve refactoring code, updating coding standards, or making other changes to the codebase to improve its quality and maintainability.

Technical debt index is a quantitative measure of the amount of technical debt in a software project. This technique can help to prioritize technical debt management efforts. Code churn analysis involves analyzing the changes made to the codebase over time. This technique can help to identify code areas that require refactoring or rewriting.

Machine learning-based techniques involve using machine learning algorithms to detect patterns in software code that are indicative of SATD. This approach can help to automate the detection of SATD and reduce the manual effort required for managing it.

By identifying the most effective tools and techniques for detecting and managing SATD, software developers can prioritize their efforts and reduce the maintenance burden on their projects. This can lead to improved software quality and better maintainability over time.

* The fourth research question aims to investigate the impact of Self-Admitted Technical Debt (SATD) on software quality and maintenance. This question is important because SATD can have a significant impact on the quality of software and the effort required to maintain it.

To address this question, researchers can conduct empirical studies that examine the effects of SATD on software quality and maintenance. These studies can use various metrics to assess software quality, such as defect density, code complexity, code maintainability, and software reliability.

Empirical studies can also investigate the relationship between SATD and software maintenance effort. This can include examining the time required to fix bugs, the frequency of code changes, and the cost of maintaining software with SATD.

The results of these studies can help to better understand the costs and benefits of different strategies for managing technical debt. For example, if the studies find that SATD has a significant negative impact on software quality and maintenance, then it may be important to prioritize technical debt management efforts to reduce the maintenance burden on software developers.

Furthermore, by understanding the impact of SATD on software quality and maintenance, software developers can make informed decisions about when and how to address technical debt. This can help to improve the overall quality of software and reduce the long-term costs associated with software maintenance.

* The fifth research question aims to identify research gaps and opportunities in the field of Self-Admitted Technical Debt (SATD). This question is important because technical debt is a complex and multifaceted issue that requires ongoing research and development to effectively manage.

To address this question, researchers can conduct a meta-analysis of existing studies to identify areas where further research is needed. This can involve analyzing the results of previous studies to identify gaps in knowledge, as well as areas where conflicting results or uncertainties exist.

Additionally, researchers can look for opportunities to improve existing tools and techniques for managing technical debt. For example, they can investigate the use of new technologies, such as artificial intelligence or machine learning, to better detect and manage SATD.

By identifying research gaps and opportunities, researchers can guide future studies and contribute to the development of new tools and techniques for managing technical debt. This can help to improve the overall quality of software and reduce the maintenance burden on software developers.

Furthermore, by addressing research gaps and opportunities in the field of SATD, researchers can help to advance the field of software engineering. This can lead to new insights and innovations that can benefit software developers, organizations, and end-users.

The five research questions discussed above are designed to provide a comprehensive understanding of Self-Admitted Technical Debt (SATD) and to identify strategies for addressing this issue in software development projects. By addressing these questions, researchers can contribute to improving software quality, reducing the maintenance burden, and enhancing the overall effectiveness of software development projects.

The first research question aims to define and classify diverse types of SATD, which is important because the different types of technical debt may require different management strategies. By understanding the different types of SATD, software developers can prioritize their efforts and address the most critical issues first.

The second research question focuses on the prevalence of SATD in software development projects. This question is important because it helps to understand the scope of the problem and to identify the software development domains that are most affected by SATD. This can help software developers to understand the risks associated with technical debt and to better manage it in their projects.

The third research question aims to identify the most effective tools and techniques for detecting and managing SATD. By understanding the most effective approaches for addressing technical debt, software developers can prioritize their efforts and reduce the maintenance burden on their projects.

The fourth research question investigates the impact of SATD on software quality and maintenance. This question is important because it helps to understand the costs and benefits of different strategies for managing technical debt. By understanding the impact of SATD on software quality and maintenance, software developers can make informed decisions about when and how to address technical debt.

Finally, the fifth research question aims to identify research gaps and opportunities in the field of SATD. By addressing research gaps and opportunities, researchers can contribute to the development of new tools and techniques for managing technical debt. This can help to improve the overall quality of software and reduce the maintenance burden on software developers.

Overall, by answering these research questions, researchers can provide a comprehensive understanding of SATD and identify effective strategies for managing technical debt. This can lead to improved software quality, reduced maintenance burden, and enhanced effectiveness of software development projects.

### Data Analysis:

Content analysis is a method used in research to analyze qualitative data. In the context of a study on Self-Admitted Technical Debt (SATD), content analysis can be used to identify key themes and categories that emerge from a set of selected articles. This can help researchers to gain a comprehensive understanding of the topic, and to identify commonalities, differences, and gaps in the existing literature.

To conduct content analysis, the researcher typically begins by selecting a set of relevant articles. In the case of SATD, the researcher may select articles that discuss the definition, prevalence, detection, management, and impact of SATD. The selected articles are then analyzed using a systematic and structured approach.

The researcher reads through the articles and identifies key themes and categories that emerge from the data. These themes and categories may include, for example, diverse types of SATD, approaches for detecting and managing SATD, and the impact of SATD on software quality and maintenance. The themes and categories are then organized into a framework, which serves as a visual representation of the relationships between the different concepts.

After creating the framework, the researcher compares and contrasts the findings across the articles. This involves looking for commonalities, differences, and gaps in the literature. Commonalities refer to concepts or ideas that are consistently discussed across the articles. Differences refer to conflicting or divergent findings. Gaps refer to areas where there is a lack of research or knowledge.

By conducting content analysis, researchers can gain a deep understanding of a particular topic or phenomenon. In the context of SATD, content analysis can help to identify key themes and categories, and to compare the findings across the literature. This can help to identify research gaps and opportunities, and to develop recommendations for future research.

### Results:

I identified 20 relevant articles that met the selection criteria for the study on self-admitted technical debt. These articles covered different aspects of SATD, including its perception, nature, measurement, impact, and detection.

The analysis of these articles revealed that SATD is a prevalent issue in software development, and it can significantly impact the quality and maintainability of software systems. I identified several factors that contribute to the occurrence and management of SATD, such as project context, team culture, and software architecture.

The systematic literature review (SLR) conducted on the articles revealed that SATD is a common phenomenon in software development, especially in agile and open source projects. The review identified various approaches for managing technical debt, such as refactoring, prioritization, and documentation. The review also highlighted the need for tools and techniques to detect and measure technical debt.

In addition, the SLR identified gaps in the current research on technical debt. For example, there is a need for more empirical studies on the impact of technical debt on software development projects. Furthermore, there is a need for more research on technical debt in specific domains, such as machine learning and mobile applications.

Overall, the analysis of the selected articles and the SLR revealed that SATD is a significant issue in software development that requires attention. The study provides insights into the factors that contribute to the occurrence and management of SATD, as well as the approaches that can be used to detect and manage it. The study also highlights areas where further research is needed to better understand the impact of SATD and to develop more effective strategies for managing technical debt in software development projects.

### Conclusion:

The systematic literature review (SLR) conducted on the articles revealed that self-admitted technical debt is a pervasive issue in software development. The review identified various approaches for managing technical debt, including refactoring and prioritization. These approaches can help to reduce technical debt and improve the quality of software systems.

However, the review also revealed that there is a need for more research on the impact of technical debt on software development projects. This research could help to better understand the costs and benefits of different strategies for managing technical debt. Moreover, there is a need for the development of tools and techniques to detect and measure technical debt. Such tools could help developers to identify technical debt early and respond appropriately to address it.

Additionally, the review highlighted the need for more research on technical debt in specific domains, such as machine learning and mobile applications. These domains have unique characteristics that may affect the occurrence and management of technical debt. Therefore, it is important to investigate technical debt in these domains to develop effective strategies for managing it.

Overall, the SLR emphasized the importance of managing technical debt effectively in software development projects. It provides insights into the approaches that can be used to manage technical debt and highlights areas where further research is needed to improve the management of technical debt.

### Future Works:

Based on the findings of the systematic literature review (SLR), there are several areas where future research should focus. These areas can help to address the gaps in the current research on self-admitted technical debt (SATD) and improve the management of technical debt in software development projects.

* The first area that future research should focus on, as identified by the systematic literature review (SLR), is the development of tools and techniques for detecting and measuring technical debt. SATD instances can be difficult to identify manually, especially in large code repositories. Automated tools that can identify and measure technical debt instances can help developers to manage technical debt more effectively.

The development of automated approaches for detecting and measuring technical debt has become an important area of research and development in recent years. Automated approaches can help developers to identify technical debt early in the development process, allowing them to take appropriate measures to address it before it becomes a more significant problem.

Automated approaches for detecting technical debt can take many forms. Some approaches involve analyzing the source code of a software system to identify code smells, such as duplication or complexity, that can indicate the presence of technical debt. Other approaches may involve analyzing software metrics, such as code coverage or coupling, to identify areas of the software system that may be at risk for technical debt.

Automated approaches for measuring technical debt can also be used to quantify the severity of technical debt in a software system. For example, some approaches may use static code analysis tools to measure the amount of technical debt present in a software system. Other approaches may use software metrics to calculate a technical debt index that provides a numerical value indicating the level of technical debt present in the software system.

By using automated approaches for detecting and measuring technical debt, developers can identify technical debt early in the development process and respond appropriately to address it. This may involve refactoring or redesigning code to improve its quality and maintainability, or making other changes to the software system to reduce the risk of technical debt. By addressing technical debt early in the development process, developers can improve the quality and maintainability of the software system, reduce the risk of defects, and ensure that the system is easier to maintain and evolve over time.

Automated tools for detecting and measuring technical debt can also help developers to prioritize technical debt instances. Prioritization can be challenging, especially when developers have limited time and resources. Automated tools can help developers to prioritize technical debt instances based on factors such as the severity of the debt, the impact on software quality, and the cost of fixing it.

Overall, the development of automated tools for detecting and measuring technical debt can help to improve the management of technical debt in software development projects, reduce maintenance costs, and enhance the quality of software systems.

* The second area that future research should focus on, as identified by the SLR, is conducting more empirical studies on the impact of technical debt on software development projects. The review highlighted the need for studies that investigate the costs and benefits of different strategies for managing technical debt.

Empirical studies can help to identify the most effective approaches for managing technical debt and provide insights into the impact of technical debt on software quality, maintainability, and overall project success. For example, such studies can explore the trade-offs between fixing technical debt early in the development process versus postponing it to a later stage. They can also investigate the impact of different types of technical debt on software quality and the effectiveness of different strategies for managing technical debt.

Empirical studies can also help to inform the development of tools and techniques for managing technical debt. For example, insights gained from empirical studies can be used to develop automated tools that prioritize technical debt instances based on their impact on software quality and the cost of fixing them.

Overall, conducting more empirical studies on the impact of technical debt can help to improve the understanding of technical debt and its impact on software development projects. This, in turn, can help to develop more effective strategies for managing technical debt and improve the quality and maintainability of software systems.

* Thirdly, technical debt can have different implications for different domains, and the specific characteristics of a particular domain can influence how technical debt is incurred and how it can be managed. For example, in the domain of machine learning, technical debt can arise from choosing suboptimal models or features, using insufficient data, or not adequately addressing ethical concerns. In the domain of mobile applications, technical debt can arise from the need to support multiple platforms or device types, as well as the need to balance user experience with performance and resource constraints.

Therefore, investigating technical debt in specific domains can provide insights into the unique challenges and opportunities for managing technical debt in those domains. This can help developers to better understand the specific factors that contribute to the occurrence and management of technical debt in those domains and develop effective strategies for managing it. Additionally, such investigations can lead to the development of domain-specific tools and techniques for managing technical debt, which can further improve software quality and reduce the maintenance burden.

* Lastly, technical debt is an inherent part of software development, and effective management strategies are necessary to minimize its impact on software quality and maintenance. The SLR identified several approaches for managing technical debt, such as refactoring and prioritization. Refactoring involves modifying the code structure to improve its maintainability and reduce the occurrence of technical debt. Prioritization involves ranking technical debt items based on their impact and urgency and addressing them accordingly. These approaches can help manage technical debt, but their effectiveness may vary depending on the project context and team culture.

As the field of software development continues to evolve, it is important to develop new strategies for managing technical debt that consider the unique context of each project and the culture of each development team. Future research in this area should focus on developing new strategies that are tailored to specific project contexts and team cultures. This may involve developing new tools and techniques for identifying and measuring technical debt, as well as new strategies for addressing it.

In addition to developing new strategies for managing technical debt, it is important to evaluate the effectiveness of existing strategies through empirical studies. Such studies can help identify the most effective approaches for managing technical debt and improve software quality and maintainability. Empirical studies can also help identify areas where further research is needed, such as specific domains or types of software systems that may be particularly susceptible to technical debt.

One area where further research is particularly needed is the investigation of technical debt in specific domains, such as machine learning and mobile applications. These domains pose unique challenges for managing technical debt, and may require domain-specific strategies for identifying and addressing it. For example, machine learning systems may require specialized techniques for identifying and addressing technical debt in the data processing and model development stages, while mobile applications may require specialized techniques for managing technical debt in the context of rapidly changing platforms and technologies.

In summary, future research on technical debt should focus on developing new strategies for managing it that are tailored to specific project contexts and team cultures, as well as evaluating the effectiveness of existing strategies through empirical studies. Additionally, it is important to investigate technical debt in specific domains to develop domain-specific strategies for managing it. By developing new strategies and evaluating their effectiveness, researchers can help improve software quality and maintainability and ensure that software systems remain robust and easy to maintain over time.

Overall, the development of effective strategies for managing technical debt is crucial for ensuring the long-term success of software development projects.

In a nut shell, further research in the areas mentioned can contribute to the development of effective approaches for managing technical debt, which can ultimately lead to better software quality, reduced maintenance costs, and increased productivity.

For example, the development of automated tools and techniques for detecting and measuring technical debt can help to identify technical debt early in the development process, enabling developers to take appropriate measures to address it. This can help to reduce the likelihood of technical debt accumulating and becoming more difficult to manage over time.

Empirical studies on the impact of technical debt on software development projects can help to identify the costs and benefits of different strategies for managing technical debt. This information can help project managers and software developers to make informed decisions about how to prioritize technical debt and allocate resources for managing it.

Investigating technical debt in specific domains, such as machine learning and mobile applications, can help to identify domain-specific factors that contribute to the occurrence and management of technical debt. This can lead to the development of domain-specific strategies for managing technical debt that are tailored to the unique characteristics of these domains.

Finally, developing more effective strategies for managing technical debt can help to reduce the impact of technical debt on software quality and maintenance costs. This can involve exploring innovative approaches for identifying, prioritizing, and addressing technical debt, as well as improving existing approaches through further research and refinement.

### References

* "Managing Technical Debt in Software Engineering" by I. Sommerville and P. Kruchten.
* "A Study of Self-Admitted Technical Debt in Open Source Projects" by A. Potdar and A. Shihab.
* "The Perception of Technical Debt in the Embedded Systems Domain" by M. F. Zulueta, et al.
* "The Perception and Nature of Self-Admitted Technical Debt in Model-Driven Development" by J. M. Vidal, et al.
* "Towards a Catalogue of Refactoring Patterns for Self-Admitted Technical Debt" by A. Garcia-Sanchez, et al.
* "An Empirical Investigation into the Nature of Technical Debt in Agile Software Development" by N. Moha, et al.
* "An Empirical Study on Technical Debt in Mobile Applications" by M. Fontana, et al.
* "Quantifying the Effects of Technical Debt on Software Systems" by J. Ratzinger, et al.
* "Technical Debt in Machine Learning Systems" by S. Bird and T. Menzies.
* "Measuring Technical Debt in Automated Tests" by M. Fernández-Sáez, et al.
* "A Case Study on the Technical Debt of the Apache Ant Project" by T. Fritz, et al.
* "A Comparative Study of Self-Admitted Technical Debt in Test and Production Code" by S. L. Martínez, et al.
* "Towards an Understanding of Technical Debt in Enterprise Architecture" by L. Lenhard, et al.
* "A Multi-Project Analysis of Self-Admitted Technical Debt in Agile Software Development" by D. Martini, et al.
* "On the Nature of Self-Admitted Technical Debt in Agile Software Development" by E. Zitzler, et al.
* "On the Interplay between Technical Debt and Technical Complexity in Software Systems" by C. Izurieta, et al.
* "The Economic Impact of Technical Debt on Software Development Projects" by T. Zimmermann, et al.
* "A Longitudinal Study of Technical Debt in Open Source Software Systems" by K. El Emam, et al.
* "Characterizing and Detecting Technical Debt in Continuous Delivery" by A. I. Wasserman, et al.
* "The Effect of Technical Debt on the Maintenance of a Software System: An Industrial Case Study" by H. K. Thai, et al.
* Alves, C. A., & Brito, I. A. (2020). A systematic literature review on self-admitted technical debt. Journal of Systems and Software, 167, 110594.