

**UNIVERSITY OF TARTU**

Institute of Computer Science

Software Engineering Curriculum

**Kamil Aliyev**

# **Securing Quality in the Software Development Process of Financial Systems**

**Master's Thesis (30 ECTS)**

Supervisor: Fredrik Payman Milani

Tartu 2025

## Securing Quality in the Software Development Process of Financial Systems

**Abstract:** The financial industry is exposed to different kinds of risk, such as operational and reputational. The financial systems should be reliable, performant, and audit-ready to minimize these risks. The way that these risks are addressed and requirements met is by ensuring the quality of these systems. In addition to the risks, the rapid evolution of financial systems adds complexity to maintaining software quality. This research aims to explore the question of how software quality is prioritized and implemented in such systems. Research utilizes systematic literature review and expert interview methods to explore the topic. The systematic literature review aimed to capture the academic contributions on quality aspects, metrics, and tools. On the other hand, the interview aimed to highlight how the quality is managed and maintained practically in the systems. Based on the results, a structured framework that can guide field practitioners and researchers was developed. The framework is developed in three dimensions, providing layered guidance on the quality aspects, how to measure them, and how to implement them in the different software lifecycle stages.

**Keywords:** Software Quality Assurance, Financial Industry, Compliance, Automation, Systematic Literature Review, Interviews, Quality Metrics

**CERC: P170**

## Tarkvaraarenduse protsessi kvaliteedi tagamine finantssüsteemides

**Lühikokkuvõte:** Finantssektor puutub kokku erinevat tüüpi riskidega, sealhulgas tegevus- ja mainekahjuga. Selliste riskide vähendamiseks peavad finantssüsteemid olema töökindlad, suure jõudlusega ning auditeerimisvalmidusega. Nende riskide maandamine ja nõuete täitmine saavutatakse süsteemide kvaliteedi tagamise kaudu. Lisaks riskidele muudab finantssüsteemide kiire areng tarkvarakvaliteedi säilitamise keerukamaks. Käesoleva uurimuse eesmärk on uurida, kuidas tarkvarakvaliteeti sellistes süsteemides prioriseeritakse ja rakendatakse. Uurimistöö kasutab teemaga tutvumiseks süstemaatilise kirjanduse ülevaate ja eksperdiintervjuude meetodeid. Süstemaatiline kirjanduse ülevaade on suunatud akadeemiliste käsitluste kogumisele kvaliteedi aspektide, mõõdikute ja tööriistade kohta. Intervjuude eesmärk on esile tuua, kuidas kvaliteeti süsteemides praktiliselt hallatakse ja säilitatakse. Tulemuste põhjal töötati välja struktureeritud raamistik, mis võib olla juhiseks valdkonna praktikutele ja teadlastele. Raamistik koosneb kolmest dimensioonist, pakkudes kihelist juhendamist kvaliteediaspektide määratlemiseks, nende mõõtmiseks ning rakendamiseks tarkvara elutsükli eri etappides.

**Märksõnad:** tarkvarakvaliteedi tagamine, finantssektor, vastavusnõuded, automatiseerimine, süstemaatiline kirjanduse ülevaade, intervjuud, kvaliteedimõõdikud

# Table of Contents

1. Introduction.....	5
2. Background.....	7
2.1 Evolution of Software Quality in Finance.....	7
2.2 Software Quality Challenges in the Financial Sector and Similar Industries.....	8
2.3 Tools, Technologies, and Frameworks in Practice.....	9
2.4 Related Work.....	9
3. Methodology.....	11
3.1 Research Design.....	11
3.2 Research Questions.....	12
3.3 Systematic Literature Review.....	13
3.3.1 Search Strategy.....	13
3.3.2 Paper Selection Criteria.....	14
3.3.3 Paper Selection Procedure.....	15
3.3.5 Data Extraction Strategy.....	17
3.3.6 Data Analysis Strategy.....	19
3.3.7 Papers Overview.....	20
3.4 Interview Study.....	21
3.4.1 Data Collection.....	21
3.4.2 Data Analysis.....	24
4. Results.....	27
4.1 What aspects of software quality are prioritized by the financial industry? (RQ1).....	27
4.1.1 Results of the SLR.....	27
4.1.2 Results of the Interviews.....	28
4.2 What functional and non-functional software quality metrics are defined and used in different financial products? (RQ2).....	32
4.2.1 Results of the SLR.....	32
4.2.2 Results of the Interviews.....	33
4.3 What methodologies and tools are measuring and assessing software quality? (RQ3).....	36
4.3.1 Results of the SLR.....	36
4.3.2 Results of the Interviews.....	38
4.4 What are the advantages and disadvantages of implementing automated quality metrics in software development? (RQ4).....	41
4.4.1 Results of the SLR.....	41
4.4.2 Results of the Interviews.....	42
4.5 What role do automated techniques like static and dynamic code analysis play in identifying vulnerabilities in complex codebases of the financial systems? (RQ5).....	45
4.5.1 Results of the SLR.....	45

4.5.2 Results of the Interviews.....	46
4.6 What impact does the prioritization of software quality have on feature release? (RQ6).....	49
4.6.1 Results of the SLR.....	49
4.6.2 Results of the Interviews.....	50
5. Discussions.....	53
5.1 Discussion of Research Questions.....	53
5.2 Framework.....	57
5.2.1 Prioritized Quality Aspects.....	60
5.2.2 Metrics and Tooling.....	61
5.2.3 Implementation Strategies.....	62
5.3 Limitations.....	63
6. Conclusion.....	65
6.1 Future Work.....	65
References.....	66
Appendix.....	69
I. Interview Guideline.....	69
II. Consent Form.....	71
III. Interview Questions.....	73
IV. Final Tag List with Definitions for Interviews.....	75
License.....	76

# 1. Introduction

The progress of information and communication technologies (ICT) in the last twenty years has resulted in the broad adoption of such systems across domains, and the financial industry was one of them [1]. In a high-stakes environment, the success of software depends on its efficiency, and reliability. Software Quality has become a concern in financial systems because of the sensitive nature of the data used in such systems. Not meeting standards can lead to consequences such as reputational damage, financial loss, and non-compliance fines. To address this, financial organizations have prioritized testing, continuous monitoring, and improving software quality to ensure service reliability and data safety.

The financial systems tend to increase complexity over time, and it is expected that these systems will operate under load. Another factor is the regulatory pressure on financial institutions. These factors have reinforced the need for Software Quality Assurance (SQA) practices and automation. Several studies contribute by providing conceptual models and taxonomies to guide the SQA in finance. For instance, Nagy et al. [2] and Top et al. [3] propose software quality taxonomies. On the other hand, Molnar et al. [4] and Aziz et al. [5] introduce adoption and benchmarking frameworks. These works can be strategically valuable, but are abstract and do not provide implementation details. In contrast, there are few implementation-focused studies, such as Kushwaha et al. [6], Mahida et al. [7], Datar et al. [8], and Guo et al. [9]. These works provide concrete examples of how quality can be implemented using pipelines, monitoring and observability tools, and domain-specific tools in the financial domain. However, these studies take a prescriptive approach and focus on narrow quality aspects. They tend not to cover aspects of wide quality and discuss the challenges and limitations of integrating the mentioned aspects.

Even though several studies address specific quality aspects, tools, and metrics, there is a lack of understanding of how organizations prioritize the quality aspects, implement, and adapt them under real-world conditions. Consequently, there is a need for a study that explores and connects this gap regarding specific challenges these organizations face in implementing software quality measures efficiently. This gap highlights the need for an examination integrating interviews with field experts and a systematic literature survey to provide comprehensive knowledge of the strategies and obstacles encountered in financial organizations' quest for high-quality software.

The primary research objective of this thesis is to identify how software quality is being implemented in modern financial products. To address this objective, the following research questions (RQ) were formulated:

**RQ1.** What aspects of software quality are prioritized by the financial industry?

**RQ2.** What functional and non-functional software quality metrics are defined and used in different financial products?

**RQ3.** What methodologies and tools are used for measuring and assessing software quality?

**RQ4.** What are the advantages and disadvantages of implementing automated quality metrics in software development?

**RQ5.** What role do automated techniques like static and dynamic code analysis play in identifying vulnerabilities in complex codebases of financial systems?

**RQ6.** What impact does the prioritization of software quality have on feature release?

In order to understand real-world insights about software quality in modern financial products, the research will employ a multifaceted research methodology. The contributions of this thesis are achieved by conducting a systematic literature survey on the topic, conducting interviews with industry experts, conducting a thematic analysis of the answers, and presenting the results in a framework. The framework would serve as a structured knowledge instance for software engineering specialists, project managers, and business analysts to detect improvement opportunities in the projects they are working on, as well as for researchers working on elaborating on the software quality topic in their research.

The rest of the thesis is structured as follows. Section 2 provides background information on software quality in the financial sector and reviews related work, focusing on existing research studies, methodologies, and findings in software quality assurance. Section 3 describes the research methodology employed in this study, including systematic literature review, interviews, and evaluation techniques used in framework creation. In Section 4, the results of the research based on the findings for each research question are presented. Section 5 presents the discussion about the results and concludes this study.

## 2. Background

The advancement of Information Technology (IT) resulted in the broad adoption of such systems in the financial sector. This adoption process moved the software quality from a support function to a mission-critical one. This section provides a background on the evolution of software quality in the domain. The gaps in the literature, which this thesis aimed to address, will also be explored in this section.

### 2.1 Evolution of Software Quality in Finance

The financial systems were initially developed using the sequential Waterfall model. This approach was suitable for well-planned systems. However, arguably, this model does not fit the modern world's agility, where business models evolve quickly, and regulatory requirements change. The model appears to fail to support long-term projects, where new features depend on system feedback [2]. Later, Agile methodologies were developed to address this issue. Agile is believed to be a turning point in development because it promotes iterative development and stakeholder involvement. As a result, Agile is believed to be more responsive than Waterfall. DevOps was the next model where integrating development and operations teams was believed to be a solution. It was designed to have continuous integration and delivery (CI/CD). As an improvement, DevOps introduced automations and delivery pipelining. Kushwaha et al. [6] showed this shift by implementing a DevSecOps pipeline for code analysis and early vulnerability detection.

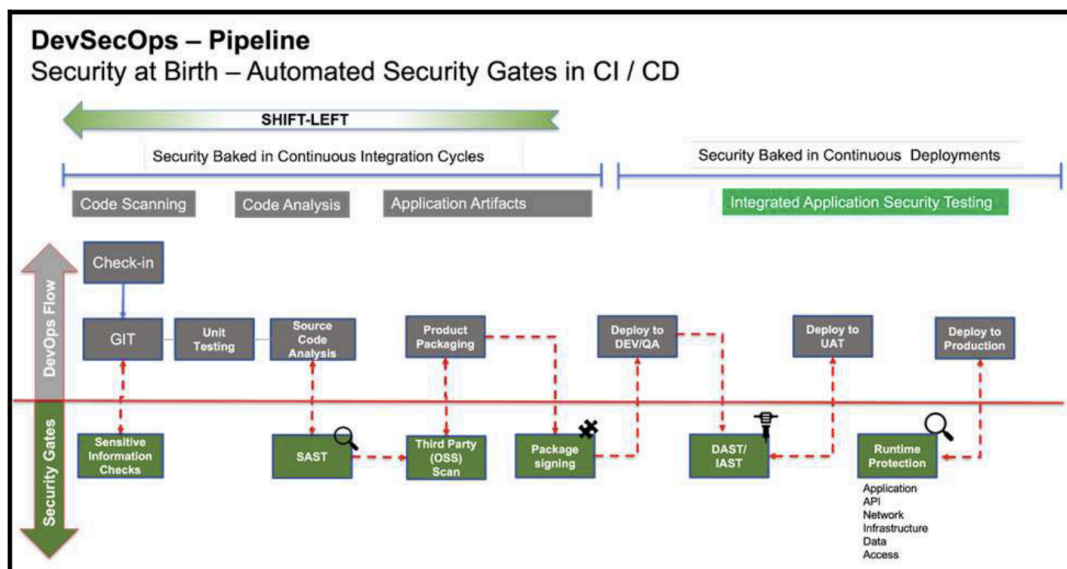


Figure 1. DevSecOps pipeline implemented by Kushwaha et al. [6]

A comparatively new trend in this evolution is counted as DevSecOps practices. DevSecOps tries to embed things like testing, embedding security, and regulatory compliance into the automated pipelines. Pardo et al. [10] wrote about how pipelines can contribute to delivery stability and rollback readiness. Mahida et al. [7] researched the implementation of observability and monitoring tools in the pipeline for real-time monitoring and compliance traceability.

The evolution appears to show industry trends that move quality assurance (QA) from checkpoints to an integrated solution to the whole development and lifecycle process.

## 2.2 Software Quality Challenges in the Financial Sector and Similar Industries

In the high-stakes industries, such as healthcare, military, and finance, quality assurance (QA) plays a key role. Such systems are expected to be available, reliable under extreme load, and to comply with regulatory requirements. For example, financial systems handle sensitive customer data and high-volume transactions in real time and maintain service continuity across global markets. Aziz et al. [5] identified service reliability as a critical driver of user adoption in internet banking systems.

One of the challenges seems to be the expectation of high reliability and fault tolerance in mission-critical systems. Hamad and Al Fayumi [11] mention that tests on system performance were conducted using large-scale data on high-availability infrastructure. The author's proposed framework, SQTL, enables scalable regression testing and dynamic test environments. The other challenges can be latency, throughput, uptime, and performance. For example, Guo et al. [9] proposed a hybrid cloud testing framework to address this issue. The proposed solution enables parallel test execution on real mobile devices in public and private clouds. This framework itself was designed to support both private and public clouds. According to the author, tests exposing sensitive user data should be tested on a private cloud to avoid breaching compliance. While Molnar et al. [4] proposed a taxonomy of software product quality metrics emphasizing attributes like correctness and maintainability, such frameworks often struggle to account for modern financial platforms' dynamic, high-throughput nature without further automation support.

Security and regulatory compliance are the other topics of concern. Financial applications must comply with the regulatory standards, such as DORA (EU) and FFIEC IT Handbook (U.S.). These regulations have strict requirements around data traceability, operation transparency, etc. Datar et al.[8] designed a platform called BVCS to automate the validation process for insurance rules. Maintaining data traceability as audit trails was one of the keys that the author focused on. While Top et al. [3] attempted to establish internal and external benchmarking repositories to



assess software quality, their work highlights the difficulty of standardizing such metrics across financial organizations with context-dependent compliance demands.

These studies reflect how domain-specific risks, regulations, and constraints are deeply entangled in the financial sector.

## 2.3 Tools, Technologies, and Frameworks in Practice

Financial systems tend to grow complex, and keeping quality high gets harder. That is why a variety of tools and frameworks are used. Some are used centrally within the organization, others are specific to a team or business group. Generally, they manage testing, monitor performance, ensure compliance, and deliver the software.

Research in this field seems to show that financial institutions focus on the tools that improve code quality and security. Some studies underscore the value of monitoring tools. For instance, Mahida et al. [7] integrated tools like Prometheus and AppDynamics into the delivery pipeline to track real-time system health, latency, and SLA breaches. These tools provide teams with fast feedback and operational stability. Some domain frameworks are explicitly built for financial systems. Panarin et al. [12] propose a Clear TH framework, which helps with recreating financial transactions and improves traceability. Hamad and Al Fayumi [11], on the other hand, introduce SCTL, a flexible testing environment designed to support high availability in complex settings. In terms of reporting and communication, Braun et al. [13] created a pipeline that uses natural language to generate and explain test results, which the authors reported to be useful for non-tech stakeholders and internal/external auditors. Similarly, Datar et al. [8] built a BVCS platform to automate form validations, generating audit-friendly logs. On the infrastructure side, Guo et al. [9] proposed a hybrid cloud testing platform that can be used for performance testing in both public and private clouds. The private cloud was there to ensure data residency and regulatory compliance. Xie et al. [14] focused on legacy banking systems, adapting a QTP-based framework for test automation.

These tools and frameworks show how financial institutions move toward integrated, automation-driven QA strategies.

## 2.4 Related Work

This research, like prior studies, aims to explore the quality assurance practices in financial systems. Its goal is to connect theoretical knowledge with the insights from the industry and deliver practical implementation guidance.

First of all, many of the studies remain at a concept level or model-driven. Works like [2], [4], [5], [15], [16] provide valuable insights into quality metrics and acceptance models. However, they often stop addressing how these outputs can be implemented in a business-as-usual environment. For example, Nagy et al.[2] use a static analysis approach in data applications to measure quality. The paper focuses on identifying quality bottlenecks without implementation. Molnar et al.[4] in their paper study software product quality metrics. This paper is a retrospective analysis rather than a guiding practical decision-making tool that can be used during implementation. Aziz et al. [5] examine the acceptance of internet banking app from the user perspective, yet the paper itself does not provide insights into the implementation of this testing in development. Ciancarini et al. [15] propose an ontology of software quality attributes, but from a theoretical perspective without examples of integration in the QA process. Similarly, Sirshar et al. [16] review SQA in high-stakes systems, but still do not address sector-specific quality assurance practices and their tradeoff.

In contrast, more recent papers like [7], [12], [17], [18] are implementation-focused. These papers demonstrate practical strategies for QA in financial settings, but they are often limited to isolated case studies, each focusing on some domain-specific implementation. For example, Mahida et al. [7] discusses monitoring and observability tools, integrated into a CI/CD pipeline. The study shows how it can improve traceability and compliance, but remains limited to the deployment and does not cover other aspects of development. Akin et al. [17] reported the transition from manual to automated testing in a bank's mobile application. Selenium and Jenkins were used in this transition; however, the paper focuses on the implementation process only and lacks details on how QA aspects were prioritized and how they affected the overall release process. Panarin et al. [12] presented ClearTH, a domain-specific test automation framework. The paper discusses the benefits of execution time but lacks details on the effects on overall organizational testing practices. Buchgeher et al. [18] introduce an architecture-based testing approach for enterprise applications in the paper. However, the paper focuses on architecture; it underscores the benefits of code modularity but lacks details of the bigger picture.

This thesis combines a quantitative Systematic Literature Review (SLR) with a qualitative interview study. The research goal of the paper aims to uncover what practices exist in the domain, how and why they are implemented, and what impact they have on the software quality of financial systems.

## 3. Methodology

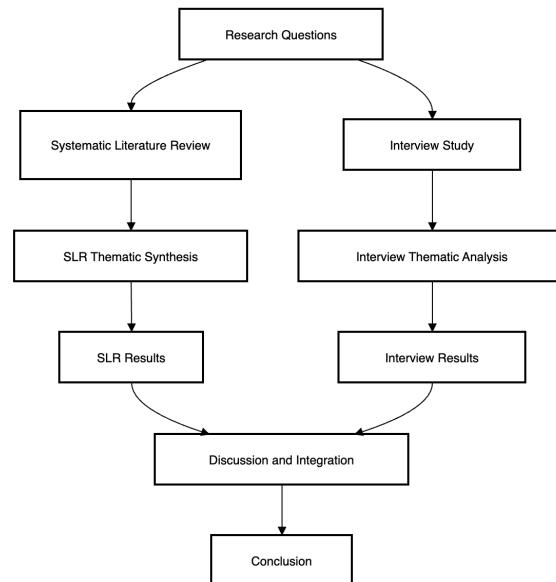
### 3.1 Research Design

This research focuses on the current state of quality assurance in software development processes, including possible enhancements to financial services. Therefore, this section presents the methods and procedures by which this research was conducted. The research will employ both qualitative and quantitative methodologies.

For the quantitative part, we will conduct a systematic literature review that will be executed with the guidelines provided by Kitchenham [19]. A well-executed literature review will include a systematic analysis of current research. It will contribute to clarifying the study's objectives, as well as providing relevant trends, gaps, and best practices in QA procedures.

The qualitative part will be covered by the semi-structured interview with the field experts to enhance comprehension of the context-specific information. That will contribute to the real-world applicability. The interviews were designed following qualitative research guidance provided by Adeoye-Olatunde and Olenik [20]. The interviews will involve individuals with considerable experience in QA techniques in the financial industry to dive deep into the field with a more practical view.

Each method was analyzed independently, and its results were later synthesized jointly as described in Section 3.6.



*Figure 2: Overview of the research*

## 3.2 Research Questions

Several research questions represent the goal of the study. The following research questions (RQ) are developed based on the identified gap:

### **RQ1: What aspects of software quality are prioritized by the financial industry?**

Understanding the reasons and priorities guiding the decisions can provide insights into how the financial industry handles risk management and system stability. This research question examines the need for software quality within the financial industry due to its high-risk nature.

### **RQ2: What functional and non-functional software quality metrics are defined and used in different financial products?**

This research question investigates the adaptation of software quality metrics across various financial products. It can help with determining a product-specific quality model in development processes by investigating the quality standards concerning product functionality, regulatory demands, and non-functional requirements.

### **RQ3. What methodologies and tools are measuring and assessing software quality?**

This research question aims to find the methods and tools for software quality evaluation. The objective also aims to determine the strengths and limitations of various quality tools, considering aspects like accuracy, scalability, and workflow integration.

### **RQ4: What are the advantages and disadvantages of implementing automated quality metrics in software development?**

Automated quality metrics have the potential to increase efficiency, and the financial sector requires reliable and high-quality software to reduce risks and adhere to regulatory requirements. This research question aims to delve deeply into the issues surrounding such automation's efficacy and possible drawbacks.

### **RQ5: What role do automated techniques like static and dynamic code analysis play in identifying vulnerabilities in complex codebases of the financial systems?**

This research question aims to assess the efficacy of automated quality analysis techniques in detecting codebase vulnerabilities. It aims to explore whether automated tools and tests can be used to detect coding errors, logical errors, and vulnerabilities.

## **RQ6. What impact does the prioritization of software quality have on feature release?**

This research question aims to determine how software quality prioritization affects feature development cycles, particularly in light of iterative release models. The investigation additionally examines how putting software quality first affects the release process's overall effectiveness and success, offering insights from the choices made by high-performing software development teams.

### **3.3 Systematic Literature Review**

This section presents one of the methods used to achieve this thesis's results, the systematic literature review (SLR). The SLR was conducted based on the Brereton et al. [21] and Kitchenham [19] guidelines. The review method was chosen to synthesize existing evidence, identify trends, and clarify gaps in the current body of knowledge.

This SLR comprises three main stages: planning, conducting, and reporting the results. The planning phase involves identifying the need for a review, formulating motivation, and developing and evaluating the review protocol. The conduction phase involves identifying studies, selecting primary papers, extracting data, and synthesizing data. The reporting phase consists of defining how the results will be disseminated and compiling a final report, including the synthesis and evaluation of results [19].

#### **3.3.1 Search Strategy**

The search strategy practiced in the literature review part is based on the guidelines and recommendations provided in these papers [11], [12], [22], [23], [24]. In order to ensure the coverage of all relevant papers, the search strategy was conducted. The following search string was generated to gather research papers from several electronic databases:

*(( "software quality" OR "software testing" OR "quality metrics" OR "software assurance" ) AND ( "financial software" OR "banking" OR "financial systems" ) AND ( "automation" OR "DevOps" OR "static analysis" ) )*

The list of terms used in the search string is the following:

- (1) *"software quality," "software testing," "quality metrics," "software assurance"*

These terms are included as one of the core domains of the research, as this is a crucial term, as the paper aims to assess and enhance the quality of software systems, particularly testing frameworks and techniques.

(2) *"financial software," "banking," "financial systems"*

These terms specify the domain context and limit assessment to studies conducted within or specifically related to the financial sector.

(3) *"automation," "DevOps," "static analysis"*

These terms exemplify modern engineering methodologies that influence software quality, especially via the automation of testing, integration, deployment, and code analysis.

The search string was executed in the electronic databases selected based on coverage of papers specific to the computer science field. The second limit of the database selection was that the databases needed to have free access rights within the university domain. The list of used databases is the following:

(1) IEEE Xplore

(2) Scopus

Source	Number of papers
IEEE Explore	27
Scopus	237
<b>Total</b>	<b>264</b>

*Table 1. The number of papers identified per source*

The results of the execution of the search string on selected electronic databases are shown in Table 1. The search results are listed and collected under a single list with 264 publications, with 2 duplicates between databases.

### 3.3.2 Paper Selection Criteria

The paper selection is the process of capturing the relevant, up-to-date, and sufficient papers that provide vital information to address the research questions. The following inclusion and exclusion criteria were used in the paper selection process:

(1) Inclusion Criteria (IC):

- 1. IC1. Does the paper discuss software quality in the financial sector?** This criterion aims to focus on the intended research area and helps filter out unrelated studies. This inclusion criterion is chosen to filter out papers that mention the

financial industry, but are not focused on it. The research requires us to look at only papers specifically scoped to software quality in the financial sector.

2. **IC2. Does the paper present or review software quality methods, tools, metrics, or frameworks?** This criterion assists in selecting resources that either present or review software quality methodologies, which directly motivates the study aimed at comprehending and enhancing software quality practices across the financial services industry.

(2) Exclusion Criteria (EC):

1. **EC1. Is the paper accessible?** Papers that can be accessed through the University of Tartu provided subscription to the digital libraries or are available on the Internet in a free-access format are considered accessible.
2. **EC2. Is the paper written in English?** Studies not published in English are removed from the analysis, as this limits the potential for broad discussion on results within the scientific community and hinders the evaluation process.
3. **EC3. Is the paper written in scientific paper format?** Ensuring that the papers are in a recognized format is vital for maintaining the quality of the research. Papers in the format of journal publications, conference papers, and book chapters satisfy the requirement of a scientific publication and will be included.
4. **EC4. Is the paper published in the last 10 years?** Ensuring that the papers are published after 2015. Papers published earlier are unlikely to describe up-to-date data about software quality in industry

### 3.3.3 Paper Selection Procedure

Based on the paper selection criteria, the following selection procedure was executed. The procedure is executed as follows: First, filter papers by publication year, filter papers by format, and filter papers by the paper content.

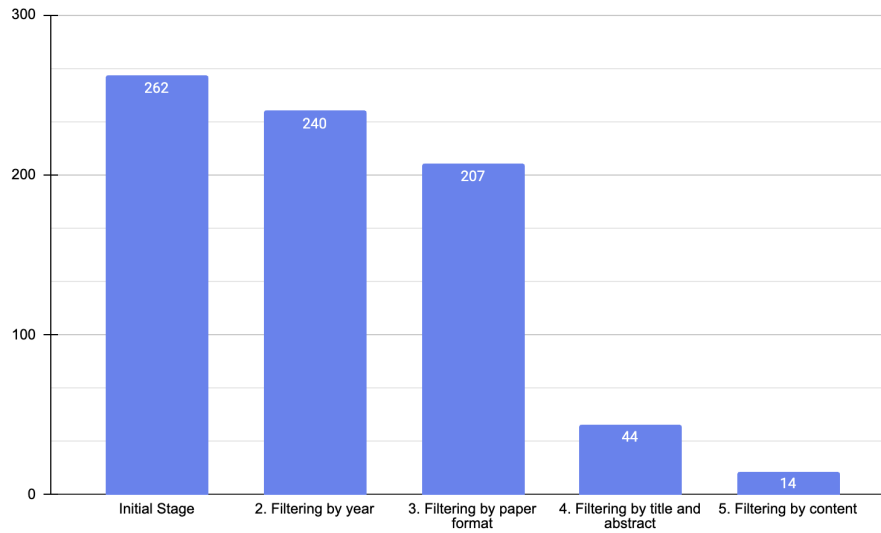
Stage	Number of identified papers	Total papers left
Primary search results	262	
Filtering by publication year	22	240
Filtering by paper format	33	207
Filtering by paper title and abstract	163	44

Filtering by reading the full paper	29	14
-------------------------------------	----	----

*Table 2. The results of filtering*

- (1) **Filtering by publication year.** Only papers published within the last 10 years were included, based on the assumption that methodologies and tools evolve rapidly. This step reduced the initial pool from 262 to 240 papers. (EC4)
- (2) **Filtering by paper format.** A filtering step was conducted to ensure that only publications with recognized scientific formats were included. Papers were retained only if they belonged to standard academic types such as journal articles, conference papers, book chapters, and publications from reputable sources like IEEE Journals, IEEE Conferences. Review papers, editorials, workshop summaries, and other non-scientific or informal publication formats were excluded. This filtering step ensured methodological rigor and relevance for the subsequent analysis. (EC3)
- (3) **Filtering by paper title and abstract.** Evaluate the relevance of each paper's title and abstract against the predefined inclusion criteria (IC1 and IC2). Papers whose titles and abstracts indicated no alignment with the research objectives—particularly those unrelated to software quality practices or the financial industry—were excluded. If a paper's relevance could not be determined based on the title alone, it was retained for further assessment during the abstract review phase. As a result of this step, 164 papers were excluded, leaving 44 papers for the next screening stage.
- (4) **Filtering by the paper content.** In the final filtering stage, the full text of each paper was reviewed to verify its contribution to the research questions and ensure it met the methodological and topical relevance defined in the inclusion criteria. Special attention was given to the presence of quality measurement techniques, automation practices, and financial domain applicability. Only papers that offered explicit insights into quality metrics, challenges, or improvement strategies within financial software systems were retained. This step resulted in 14 primary papers being selected for data extraction and analysis.





*Figure 3. The Paper count per filtering stage*

### 3.3.5 Data Extraction Strategy

A structured data extraction form was created to collect relevant findings from the chosen studies methodically. Following guidance from Fink [24], Okoli [23], and Levy and Ellis [22], the use of a standardized extraction form ensured a consistent and unbiased approach to collecting data, aligned with the study's objectives.

The data extraction concentrated on important thematic areas representing how software quality is safeguarded and implemented in financial technology systems, instead of strictly mapping fields one-to-one with each research question. This strategy allowed for flexibility while ensuring that all six research questions were covered.

There were four primary sections to the final extraction form:

- 1. Identification Data:** General metadata, such as the paper ID, title, authors, and year of publication, were recorded for traceability.
- 2. Study Context:** To facilitate comparison across various application settings, the domain or industry and the type of system under study were noted.
- 3. QA Contribution:** Concentrated on the aspects of each study that are related to quality, such as:

- **The quality focus area** (such as maintainability, performance, compliance),
- **Tools, methods, or frameworks** (such as static code analysis),
- **Quality Metrics** (functional metrics, non-functional metrics)
- **Scope of implementation** (such as unit-level, architectural, or organizational),
- **Involvement of automation** (if any),
- **Limitations of automation**

#### 4. Impact of Implementation and Delivery: Captured insights related to:

- **Implementation difficulties** (such as legacy systems and organizational resistance),
- **Impacts on software delivery** (such as speed, traceability, deployment improvements), and
- **Regulatory impact or compliance** (such as audit traceability and financial regulations).

All 14 selected papers were manually reviewed using the extraction form defined in Table 2. Any indefiniteness was resolved by rechecking the full text for alignment with the inclusion logic and thematic focus.

A summary of the final extraction fields is presented in Table 3.

Category	Field	Description
<b>Identification Data</b>	ID	Unique identifier for the paper
	Title	Full title of the paper
	Authors	Listed Authors
	Year	Year of publication
<b>Context</b>	Domain/Sector	Industry domain
	Type of System	Type of software or system
<b>Quality Contribution</b>	Quality Focus Area	Functional or non-functional aspects addressed
	QA Method/Tool/Framework	Tools, techniques, or approaches used to ensure software quality
	Implementation Scope	Scope of application (e.g., architecture, code-level, QA process)

	Automation Focus	Whether automation is discussed, and in what area
	Automation Effects	Benefits and drawbacks of automation
<b>Implementation and Delivery Impact</b>	Effect on Delivery	Reported influence on speed, traceability, compliance, or release performance
	Implementation Challenges	Barriers faced in implementing QA
	Compliance/Regulation Impact	QA's effect on meeting regulations or standards

*Table 3. Data Extraction Form*

### 3.3.6 Data Analysis Strategy

The next phase after Data Extraction is Data Analysis. The objective of this stage is to systematically analyze and organize the extracted information from the papers found in the SLR step, which is done by mapping the excerpts from the papers in a way that answers the study's research questions.

Given the exploratory nature of the research, a thematic synthesis approach was chosen. This method helps identify patterns, practices, and insights related to software quality in the financial domain.

The analysis was conducted by using the following steps:

- 1. The first step involved a thorough review of each entry in the extracted data.** All the fields from the data extraction form were analyzed: Quality Focus Area, Quality Approaches/Frameworks/Metrics, Implementation Scope, Automation Effects, Implementation Challenges, and Effects on Delivery. This step enables a comprehensive understanding of how quality is defined, prioritized, and experienced in different phases of the SLR results.
- 2. Initial Coding and Categorization:** In this step, commonalities across the information discussed in the paper were noted and grouped. This step enabled the extraction of the key recurring themes.
- 3. Thematic Aggregation:** In this step, extracted data points from the Data Extraction Table were categorized and aggregated into themes.

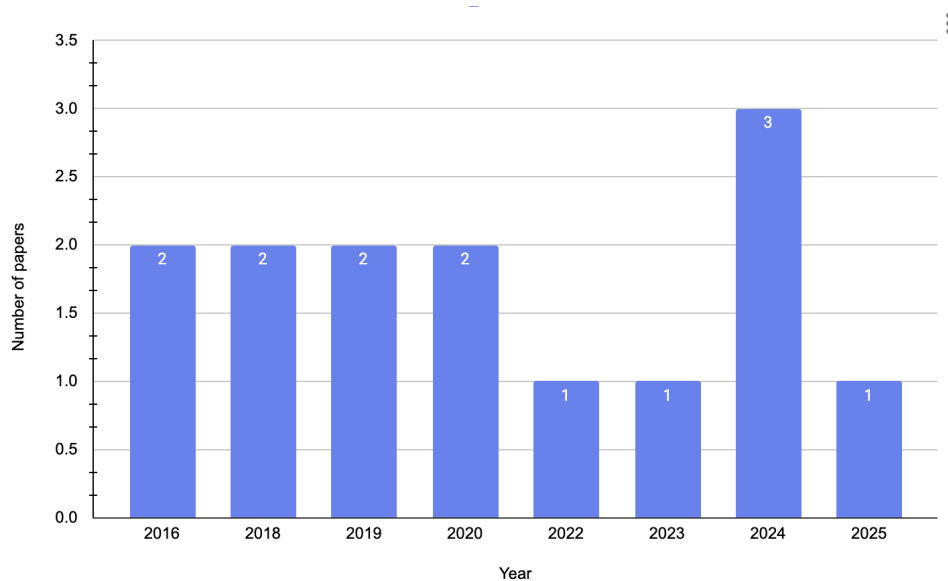
**4. Cross-Theme Linking:** In conclusion, we explored the connections between themes.

Ultimately, these relationships between data points and themes were synthesized to form a narrative that explores how financial software systems secure quality at different levels, positions, and organizations.

The results of this step form the basis of the synthesis presented in the Results chapter. This strategy is effective because it ensures the roots of the findings are secure and well defined, while the collected data is being mapped to the research objectives of the paper

### 3.3.7 Papers Overview

This section will provide an overview of the papers included in this research. By utilizing a set of inclusion and exclusion criteria, the total number of papers eligible for extraction is 14. The graph below shows the distribution of the papers by publication year. Fewer researchers explored this topic recently (2016-2025), resulting in few published papers. In some years, no papers were published at all.



*Figure 4. Paper distribution over the publication date*

The selected studies address a broad spectrum of themes aligned with this study's research objective. Specifically, many papers have focus on:

- Software Quality Aspects such as security, compliance, and performance
- Automation Practices for delivery speed, delivery consistency, vulnerability detection, and audit readiness

- Domain-specific testing Tools and Frameworks were built for compliance purposes, cost, and time efficiency

These studies include domain-specific solutions (e.g., ClearTH, SCTL, and BVCCS) and general frameworks (e.g., DevSecOps, test automation). Collectively, the selected papers provide a foundation to address the research objectives of this paper.

### 3.4 Interview Study

This section describes the interview study conducted to explore industry professionals' views on the topic. The semi-structured interviews were conducted with professionals who directly worked and were experienced in the financial industry and could provide insights on the topic. This section is divided into two parts: Data Collection and Data Analysis.

#### 3.4.1 Data Collection

A semi-structured interview format was chosen to balance consistency across interviews with the flexibility to explore context in greater depth. This approach allows candidates to share more knowledge by using open-ended responses while enabling the interviewer to adapt questions based on the candidate's experience and domain knowledge.



*Figure 5. Overview of data collection*

#### **Interview Guidelines**

Interview questions (IQs) were designed to target specific aspects of research questions (RQs). The seven main and accompanying subquestions were developed, and the thematic domains of research objectives guided the structure of these questions. While the complete list of interview questions is included in Appendix III, the RQ–IQ mapping is presented below for transparency and traceability.

Interview Questions	RQs	Explanation
IQ1: How is the software quality of the projects planned in the early design and planning phase?	RQ1, RQ6	To explore when quality is prioritized and its influence on delivery
IQ2: What functional and non-functional quality metrics are generally used by your team?	RQ2	To identify common metrics
IQ3: What critical aspects of software quality are generally prioritized by your company?	RQ1	To understand which quality attributes are emphasized and why
IQ4: What frameworks, tools, and methodologies are most common and vital for projects in the financial industry?	RQ3	To collect data on practical tools and frameworks
IQ5: What are the key advantages and disadvantages of implementing automated quality metrics in software development?	RQ4, RQ5	To evaluate automation effects on delivery and quality efforts
IQ6: What impact does the prioritization of software quality have on future rolling?	RQ6	To understand the effects of quality, focus on the long term
IQ7: How does the organization ensure compliance with local and global regulations related to the software quality of the financial projects?	RQ1, RQ2, RQ3	To examine the effects of compliance and regulators on quality

*Table 4. Interview Questions mapping with Research Questions*

The search for interview participants and the connection process were accomplished through professional social platforms. The backgrounds of selected industrial professionals are represented in Table 5. A minimal criteria list will be followed for interview selection:

1. 2+ years of experience in the Financial Industry
2. Current or past employment in the financial sector
3. Familiarity with QA tools and methodologies
4. Effective communication skills
5. No limit to geographical locations

ID	Title	Experience	Sector Subdomain
P1	Software Engineer	Five years	Fintech—App development
P2	Software Engineer	Four years	Commercial Banking—Software Delivery
P3	Test Engineer	Four years	Fintech—Payment Processing
P4	Digital Manager	Three years	Fintech—Payment Processing
P5	DevOps Engineer	Seven years	Commercial Banking—Platform Ops
P6	Tech Lead	Eight years	Commercial Banking—Data Warehouse team
P7	Senior Software Engineer in Test	Eight years	Fintech—Digital Payments
P8	Product Owner	Three years	Fintech—Mobile Finance
P9	Quant Engineer	Four years	Investment Banking – Front Office
P10	Site Reliability Engineer	Three years	Investment Banking – Platform Reliability

*Table 5. Interview candidates and their background*

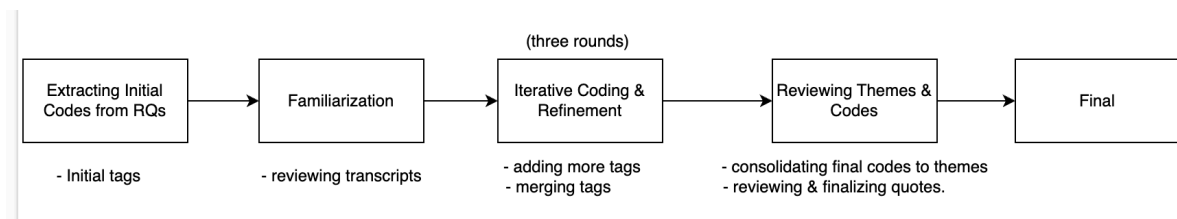
The study was seeking individuals with at least 2+ years of experience who could share their experience with various challenges and scenarios. This criterion ensured that the validated experience of QA principles has moved beyond basic understanding. Their practical knowledge and experience in the financial sector can contribute meaningful insights. Thirdly, familiarity with QA tools and methodologies indicates that the interviewees can provide detailed information on the effectiveness, advantages, and limitations according to their hands-on experience. This criterion was essential for providing absolute solutions for issues emerging from the software development lifecycle. Practical communication skills are important for interviewees to share their experience and knowledge clearly and comprehensively. This criterion facilitated the collection of high-quality responses, presented in detail, to enhance the understanding of work dynamics. During the selection of interviewees, participants from all over the world were considered. Since the thesis aims to study how QA practices are implemented in different environments, it is not limited to a specific region but covers the view of QA globally.

In addition, the following standards shall be followed to ensure the validity of the material acquired during the thematic analysis of interviews:

1. **Minimum criteria list for interviewees:** A minimum requirement list has been developed to find individuals with relevant expertise who can provide valuable feedback that will contribute to the study.
2. **The interview guideline:** The interview guideline was designed to guarantee a solid structure and ensure thematic comparability for interviews. The interview guideline is presented in Appendix I, which consists of the following:
  - a. Welcome and introduction to the research topic
  - b. Verbal consent and recording
  - c. Interview questions
  - d. Clarifying questions (if any)
  - e. Finalizing and the end of the recording
3. **Consent form:** All interviews conducted took place on Zoom and were recorded to increase the reliability of the results. Interviewees must sign a consent form, presented in Appendix II, when agreeing to interview or give verbal consent during the recording process. The interview process must comply with the interview guidelines and take 45 to 60 minutes. All recordings are deleted after transcription for processing.
4. **Verification of participants:** Feedback and verification from respondents will be collected to ensure the authenticity of their views and findings. This will ensure the project results accurately capture and honestly express all the interview findings.
5. **Semi-structured interview:** The pre-test is critical in establishing if interview questions are practical, clear, and successful in addressing their intended purpose. Feedback from the first interviewee during this test will help ensure that the changes made to the questions lead to better clarity and comprehensibility.

### 3.4.2 Data Analysis

The collected interview data were analyzed using a thematic analysis approach. The goal for thematic analysis was to extract meaningful insights regarding software quality practices in financial software development. Thematic analysis offers a flexible yet structured method for identifying, organizing, and interpreting patterns within qualitative data [25], [26], which was the main rationale behind choosing it. The overview of the data analysis is presented in Figure 6.



*Figure 6. Overview of data analysis*



The process began by developing initial codes based on the research questions. A total of 18 initial themes were generated, directly reflecting the research questions. Each research question had two or three themes. For example, the tag “Quality Planning” was derived from **RQ1**, while “Automation Limitations” and “Static Code Analysis” emerged from **RQ4** and **RQ5**, respectively.

All ten interview transcripts were reviewed to ensure familiarity with the data content. For transcribing the interviews, we used OtterAI<sup>1</sup>, and for refining and structuring, we used the ChatGPT<sup>2</sup>. The coding phase began with assigning quotes to these initial tags using Miro<sup>3</sup>. These quotes were manually highlighted and grouped into categories based on their thematic relevance. For traceability, every tagged quote included interviewee IDs.

Three iterative and manual coding rounds came after this initial tagging. Codes that overlapped or were redundant were combined in these rounds. For instance, “Defect Count” and “Bug Count” were consolidated under “Functional Metrics.” Where appropriate, new subthemes were developed, such as classifying “throughput,” “latency,” and “availability” under “Non-Functional Metrics.” Ten major themes were finally identified as a result of the refinement process. Appendix IV contains the final tag structures and definitions.

Quotes were finalized only after this thematic consolidation. Each quote was linked to its corresponding theme and marked with a strength level (low, medium, high) based on clarity, thematic fit, and relevance across participants. This structured quote set was then formatted in Microsoft Excel<sup>4</sup> and forms the foundation for the Results section.

This process ensured that each theme was clearly defined and valuable in answering the research questions.

---

<sup>1</sup> Otter.ai is a AI-powered transcription and note-taking services, used for meetings. For more information: <https://otter.ai/meeting-agent>

<sup>2</sup> ChatGPT is a generative artificial intelligence chatbot developed by OpenAI. For more information: <https://openai.com/>

<sup>3</sup> Miro is an online collaborative whiteboard platform designed for teams to brainstorm, plan, and innovate. For more information: <https://miro.com/>

<sup>4</sup> Microsoft Excel is a spreadsheet editor developed by Microsoft. It features calculation or computation capabilities, graphing tools, pivot tables. For more information: <https://excel.cloud.microsoft/en-us/>

LLM-based AI models were used in this thesis to support the writing process. Grammarly<sup>5</sup> was used to improve the clarity of text and fix grammatical mistakes. ChatGPT<sup>2</sup> was used in the research as a text refinement and improvement tool that helps to improve the structure of the written content. For example, initially, the interview excerpts were presented separately from the paragraphs, but it was decided to keep them integrated into the text itself for better readability, ChatGPT<sup>2</sup> helped with that task. During the interviews, we used a tool called OtterAI<sup>1</sup> to record and extract the transcripts of the interviews. Later, for refining and structuring the messy content of the transcripts, ChatGPT<sup>2</sup> was used. However, all analyses were manually verified, and no AI tools were used to generate research data, interview contents, or results. All sources used in the paper are based on verifiable documents and interview transcripts.

---

<sup>5</sup> Grammarly is a writing assistant software tool. It reviews the spelling, grammar, and tone. For more information: <https://www.grammarly.com/>

## 4. Results

This section provides the results of both quantitative and qualitative studies. The results are listed aligned with the RQ that they are addressing. The first RQ explores the software quality aspects of priority for the financial industry. The second RQ analyzes which metrics are defined and used in the industry. The third RQ analyzes the tooling adopted by companies to assess the quality. The fourth and fifth questions are aimed at analyzing the automation aspect of software quality by the industry, including advantages and tradeoffs. Finally, the sixth RQ aimed to examine the outcomes of all these processes mentioned in other RQs. The results of each RQ are developed in two parts. First, the results of SLR are reported, and by doing so, literature contributions are covered first. After the interview results are reported, the knowledge is expanded with field expert insights.

### 4.1 What aspects of software quality are prioritized by the financial industry? (RQ1)

This sub-section presents the results for RQ1, which aims to discover the in-use and prioritized aspects of software quality for financial systems. Both the literature and interviews had some overlapping aspects, particularly security, compliance, performance, and reliability. These aspects seem to be prioritized to minimize regulatory and operational risks.

#### 4.1.1 Results of the SLR

The systematic literature review revealed that financial industry systems emphasize that some software aspects are critical, such as **security**, **compliance**, **reliability**, **availability**, and **performance**. These categories reflect that the financial sector is sensitive to regulatory and operational risks.

**Security:** Security was among the most prioritized aspects, especially in web-based and mobile platforms. Kushwaha et al. [14] discuss a DevSecOps pipeline that uses static and dynamic codebase scanning tools (Codacy, OWASP ZAP, GitHub Advanced Security) to find vulnerabilities early in the continuous integration (CI) pipelines. Schneider et al. [27] integrated a fuzzing security technique with the Fuzino app to enhance security with dynamic testing in a mobile banking application. Other papers, such as [7] and [8], also underline how important security is in their discussions. These results show that security is a high priority in the financial sector.

**Compliance and Audit Readiness:** Compliance with financial regulations and audit readiness were mentioned and underscored in most papers. Papers like [7] and [8] describe how they can be incorporated into the development processes and improve audit readiness by making the code more traceable with better logs. Datar et al. [8] develop a validation framework that also enables automated checks for regulatory rules, both internal and external to the business. On the other hand, Mahida et al. [7] propose a different approach to validation by integrating validation processes into CI/CD pipelines and enhancing logging to ensure traceability and auditability.

**Availability, Reliability & Performance:** Several papers mentioned service availability, reliability, and performance as other key factors for financial systems. Financial systems should have low latency, high throughput, and availability under load to meet the service level objectives (SLA), and performance tests to test this. Papers like [6], [7], [9], [11], and [28] tried to improve these approaches. For instance, Guo et al. [9] offers improvement by implementing hybrid cloud testing environment in order to ease the testing process by paralilizing the tests, decreasing the cost of testing and complying with regulation by using internal cloud for keeping the sensitive data safe, author also suggests using this environment to simulate load and conduct performance testing of the software. On the other hand, Santhupadi et al. [28] highlight the need for high throughput and exemplary performance in the investment banking domain. The author proposes to use a real-time big data analytics framework to monitor the system throughput, latency, and performance. Mahida et al. [7] discuss in the paper how to make sure that the SLA goals of the team are met and improve the uptime of the systems by through implementing observability for system and application directly to CI/CD platforms of the bank, by doing that, the authors believe that the amount of SLA related incidents will decrease.

#### 4.1.2 Results of the Interviews

Based on the interviews, four criteria were repeatedly highlighted among the interviewees: **security, compliance, performance, and reliability**. These were constantly characterized as non-negotiable because they mentioned regulatory, operational, or reputational risk factors that have a significant impact. Other quality factors, such as maintainability or developer experience, have rarely been mentioned. Performance and dependability were viewed as crucial in time-sensitive and high-availability systems. At the same time, security and compliance were mentioned most frequently and were frequently connected to formal audit requirements. The participants who brought up each topic are displayed in the table below.

Participant	Security	Compliance	Performance	Availability & Reliability
P1	1	0	1	1
P2	0	1	1	1
P3	0	0	1	1
P4	1	0	0	0
P5	0	1	1	0
P6	1	1	0	1
P7	1	1	0	0
P8	1	0	0	0
P9	0	1	1	1
P10	0	0	0	0
<b>Total</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>

Table 6. Quality aspects referenced by participants in response to RQ1.

**Security:** Five participants from different backgrounds and sectors cited that security was the most important quality issue. For many, it was considered more than just a requirement; it was a foundational constraint that shapes how systems are designed, tested, and deployed.

In the financial sector, where systems deal with sensitive personal and transactional data, the consequences of a breach were described as both regulatory, reputational, and financial risk. For example, P3, a test engineer, emphasized that “*security is an extremely high priority*,” explaining that their team is taking security audits, code coverage, and fault tolerance seriously and focuses on their results. For some participants, security was a reason for choosing specific tools and frameworks. For instance, P8, a product owner, mentioned that “*security comes first because of past events and the nature of the industry*.” This mention reinforces the idea that the decision-making process was based on the need to reduce potential attack surfaces, to secure user data, and the system. For P7, a senior software engineer with 8 years of experience, highlighted that security and internal/external audit activities are closely linked. He noted that “*security and*

*data privacy are not up for discussion; they (auditors) verify logs, encryption, and permissions before publishing.”*

Overall, security was perceived as a cross-cutting quality factor influencing code review, architecture, deployment policies, and internal documentation processes.

**Compliance:** Five interviewees cited compliance, which they regarded as another non-negotiable criterion that directly influenced planning, documentation, and development procedures.

Rather than being treated as an external checklist, several participants noted that compliance shaped what “quality” meant in practice, often driving both processes and tooling choices. For instance, P6, a tech lead, framed compliance as an upstream influence on the planning. They noted, *“We have to write quality documents for compliance anyway, so that shapes what gets planned.”* By stating it, we can say that compliance dictates planning-phase decisions. P8, a product owner, added: *“Prioritizing compliance facilitates smooth audits. You prevent surprises at the end of a project.”* Similarly, P7, a senior software engineer in test, noted that audit gates have certain checks that get verified first, such as secure logging and encryption.

The above comments demonstrate how **regulatory and audit expectations** form the cornerstone of quality planning in many financial development contexts. Compliance is not a side project; it specifies what must be accomplished.

**Performance:** Four interviewees mentioned performance, typically in cases where load of the system or system responsiveness directly influenced users or business outcomes. For instance, P5, a devops engineer, highlighted that performance cannot be compromised during high demand, explained as *“performance at periods of high demand, such as wage payments, cannot be compromised. These set our service-level targets.”* Likewise, P9, a quant engineer whose role is based on real-time trading environments, emphasized that *“traders want answers right away; performance is closely related to SLOs and quick decisions.”* P2, a software engineer, added that although performance was often considered a technical dimension, it significantly influenced planning and delivery priorities: *“tracking throughput and latency for batch jobs and frontend calls. If the dashboards spike, that’s a release blocker.”*

Across these responses, performance was treated as a **measurable, business-critical aspect** of quality, not as an abstract goal, but as a threshold that must be achieved to protect trust and function.

**Availability & Reliability** were described in terms of **uptime, system stability, and fault handling**, and were mentioned by four interviewees. It was framed as a basic operational

expectation, particularly for backend or customer-critical systems. According to P3, a test engineer, pointed out that failure response and tracking are necessary by saying: *“Once the product is live, uptime and bug count matter the most.”* P2, software engineer, reaffirmed this and mentioned that observability helps detect stability problems and transparency of reliability metrics by mentioning that *“quality reporting always includes our error rates and uptime data. Everybody can see these figures.”* P10, a site reliability engineer, underlined the significance of the system behavior in recovery situations: *“Resilience is key—can a system recover quickly? Next is observability: can we identify the issue?”*. This statement demonstrates how uptime measures reliability and when the system needs a recovery.

The interviews demonstrated that **constant system availability and fast failure recovery** were essential to assessing and perceiving quality, even though not all teams used "reliability" specifically.

Name	Description	Example
<b>Security</b>	Ensuring confidentiality, integrity, and limited access to systems and data. Security is described as a fundamental requirement and is included in system architecture, deployment, and code-level practices.	Financial institutions integrate security checks early in development. One interviewee mentioned quarterly security audits, which include code coverage, fault tolerance, and encryption validation (P3). Meanwhile, the others stressed the importance of minimizing breach risk through static analysis and strict access control during infrastructure implementation (P7, P8).
<b>Compliance</b>	Following the internal governance and external regulations (e.g., PCI DSS, GDPR) that specify how software is developed, tested, and documented. Compliance requirements frequently shape the idea of quality itself.	Several interviewees noted that audit readiness and documentation were planned from the start. One explained that quality documentation exists “for compliance anyway,” which guides planning activities (P6). Another described how pre-release validations check for log completeness and encryption as part of compliance gates (P7).
<b>Performance</b>	System behavior and responsiveness under workload and pressure, including latency,	During compensation payments, when the performance targets become service-level constraints, the system stress has been described by P5. Another participant explained

	throughput, and real-time responsiveness. Business-critical processes, such as trading or batch workloads, are often related to performance.	that performance measures (e.g., reaction time) decide whether to approve a release. Performance is a top consideration in a high-speed trading environment as delays directly affect decisions (P9).
<b>Reliability</b>	System stability and consistency in both ideal and challenging situations. It covers resilience, uptime, and incident recovery skills.	Interviewees suggested monitoring incident frequency and error rates to maintain reliability (P2). Assuming "five nines" availability for backend and reporting systems was common. Fast recovery and transparency during downtime were emphasized by one interviewee (P10), while another noted that if the system is unavailable, no other measures are important (P3).

*Table 7: Summary of Quality Aspects*

## 4.2 What functional and non-functional software quality metrics are defined and used in different financial products? (RQ2)

This sub-section reports the results for RQ2 and focuses on quality metrics used in the systems. The results are structured into functional, non-functional, and compliance. Besides capturing the metrics themselves, results also cover the implementation challenges.

### 4.2.1 Results of the SLR

Even though most papers are not focused on quality metrics and have a small number of metrics mentioned, several metrics should be mentioned across different papers. They included both functional and non-functional metrics. Generally, two types of metrics were mentioned: Performance metrics and functional QA metrics.

**Performance metrics:** Mahida et al. [7] offered system health metrics as a solution by implementing via observability tools like Prometheus and AppDynamics to align system performance with SLA expectations and enable real-time issue detection. On the other hand, Guo et al. [9] recommended using CPU, resource usage for validating performance for cloud-based automated testing environments that the authors proposed. Sathupadi et al. [28] also used the system metrics like CPU and memory usage to ensure high performance and secure internet



banking. The author suggests using the custom Banknet platform to collect real-time usage data on latency spikes and resource usage for flagging anomalous transactional behaviors.

**Quality Assurance metrics:** Kushawa et al. [6] in the paper used static scanning analysis via Codacy and OWASP ZAP in order to ensure the security of the platform and improve code quality; by doing so, it indirectly can be implied that these tools report code-related QA metrics like Test Coverage, Code Smells, Security Vulnerabilities. However, Datar et al. [8] suggest a platform for automating insurance form validation and checking via the BVCS model to increase accuracy, reduce manual efforts, and support auditability. The author uses Validation Accuracy and Rule Coverage as metrics for that purpose.

#### 4.2.2 Results of the Interviews

RQ2 analyzes several metrics used to measure software quality in financial sector projects, and the challenges related to their implementations were discussed in the interview. Based on the answers, quality metrics are divided into three broad categories: **functional metrics**, **non-functional metrics**, and **compliance-driven metrics**. Participants mentioned issues related to **inconsistency**, **poor relevance**, and **challenges in standardizing metric usage among teams**. The following table lists the participants who brought up each measure and whether they discussed difficulties interpreting or implementing it.

Participant	Functional Metrics	Non-Functional Metrics	Compliance Metrics	Implementation Challenges
P1	1	1	1	1
P2	1	1	0	0
P3	1	1	1	1
P4	0	0	0	0
P5	1	1	0	1
P6	0	0	0	1
P7	1	1	0	0
P8	0	0	0	0
P9	1	1	0	1

P10	1	1	0	1
Total	7	7	2	6

Table 8. Metric Categories and Implementation Challenges referenced by participants in response to RQ2.

**Functional Metrics** were discussed by seven out of ten interviewees. The most used metrics are mentioned as **bug count**, **test coverage**, **defect density**, and **success/error rates**. These were used to measure internal code quality, regression risk, and the effectiveness of automated testing. For instance, P1, a software engineer, emphasized the role of bug count and errors as a key performance indicator by stating: *“Once the product is live, uptime and bug count matter the most.”* On the other hand, P2, a software engineer, and P7, a senior software engineer in test, indicated using **test coverage** as a standard practice. P2 mentioned that they *“look at the test coverage, especially before releases. But it doesn’t always mean quality,”* while P7 noted about dashboards to track which is monitored regularly: *“We have dashboards for coverage and test success rate. Teams track it regularly.”* P5, devops engineer, explained that the team monitors **output accuracy** as a functional metric, since *“it must match reference results within tolerance.”* P9, a quant engineer, added that they monitor *“success and error rates of operations across endpoints. It’s tied directly to quality,”* strengthening the idea of how these metrics are used for measuring the operational health of the system.

Functional metrics were widespread, but several participants acknowledged they must be interpreted carefully and supplemented by qualitative understanding.

**Non-functional metrics:** Seven participants also mentioned non-functional metrics, which are generally related to runtime behavior, such as **latency**, **availability**, **error rates**, and **resource consumption**. These were tracked using monitoring tools and tied to Service Level Objectives (SLO) or Service Level Agreements (SLA). P2, a software engineer, shared that *“we track throughput and latency for batch jobs and frontend calls. If the dashboards spike, that’s a release blocker,”* pointing out how these metrics can be gatekeepers for the feature releases. P3, test engineer, mentioned the importance of response time, latency, and performance under load. On the other hand, P10, site reliability engineer, referenced tracked metrics like mean time to recovery (MTTR) and deployment stability by noting: *“We track MTTR, deployment success rate, and resource saturation. It’s our main measure of system health.”*

These metrics were closely tied to live system behavior and, in most cases, used for real-time alerting and release gating.

**Compliance Metrics:** Only two participants (P1 and P3) directly referenced metrics linked to compliance or audit requirements. These metrics typically ensured that **regulatory obligations were traceable and met**. P1, a software engineer, noted that logging of the systems and test results *“must be clean enough for audit review,”* and they also *“store structured data to prove coverage.”* P3, test engineer, emphasized how **test mapping and documentation** were used to pass compliance checks: *“We map coverage to requirements. It’s needed for our internal audit, especially for banking projects.”*

These metrics were **less about system behavior** and more about showing compliance readiness through evidence of quality practices.

**Implementation Challenges:** Despite discussions on how widely the metrics were used, six participants expressed concerns about using and interpreting quality metrics. These included: **defining meaningful metrics, tooling and dashboard limitations, legacy infrastructure constraints, and shifting business expectations**. For example, P6, tech lead, is particularly worried about metric practicality by noting that *“we measure things, but whether they mean anything — that’s a different question.”* P3, test engineer, also added that metrics can drift or become outdated as the systems evolve: *“Some metrics are inherited from old tools or teams. They may not reflect current architecture.”* On the other hand, P10, the site reliability engineer, described the infrastructure barriers and monitoring gaps, stating, *“legacy systems don’t expose usable metrics. We rely on manual logs or guesswork in places.”*

Despite these challenges, most participants agreed that metrics were necessary, but emphasized the need for **contextual interpretation and continuous review**.

Name	Description	Example
<b>Functional Metrics</b>	Metrics related to internal code quality and behavior during development, such as defect count, test coverage, and success/error rates. These are widely used to evaluate stability before deployment and track regressions across builds.	Bug count and test success rate are tracked before releases to monitor quality (P1, P2). P5 described verifying model output accuracy for financial calculations. P9 mentioned transaction success/error rates used in API health checks.
<b>Non-Functional Metrics</b>	Runtime behavior metrics such as latency, uptime, throughput, or MTTR. These are often tied to SLAs or SLOs and monitored using observability tools across	P2 explained monitoring throughput and latency for both backend jobs and frontend requests. P10 noted the use of dashboards tracking MTTR and deployment health. P3 referenced load

	environments.	testing as a way to ensure latency requirements are met.
<b>Compliance Metrics</b>	Metrics that support regulatory requirements or audit trails, often focused on traceability, test mapping, and pre-release validation. These are used less frequently but are crucial in regulated financial institutions.	P3 described mapping test cases to functional requirements for audit purposes. P1 mentioned maintaining logs and test result outputs that could be presented during compliance review processes.
<b>Implementation Challenges</b>	Practical difficulties teams face when implementing or using metrics include unclear definitions, tooling limitations, inconsistent application, or infrastructure gaps.	P6 shared that teams often “measure things that don’t mean anything” due to a lack of context. P3 noted that some metrics persisted from legacy teams and systems. P10 explained that key systems did not expose reliable metrics due to outdated architecture.

*Table 9. Summary of Quality Metrics and Implementation Challenges*

### 4.3 What methodologies and tools are measuring and assessing software quality? (RQ3)

This sub-section reports the results for RQ3, which explores tools, methodologies, and practices in the field. The results show that different categories of tooling are used: tools for testing, observability, monitoring, and security. The tools help with the quality assurance of both code and system.

#### 4.3.1 Results of the SLR

The reviewed literature mentioned a wide variety of tools and frameworks, and also developed and tested new ones designated for quality assurance across various stages that financial software passes through. These tools range from automated testing tools, code analysis tools, observability and monitoring tools, and tools designed for the domain. These tools are generally used for automation, security, traceability, and compliance. Across the studies, the mentioned tools can be divided into three groups: Software Quality & Security, observability and monitoring, and Domain-Specific frameworks.

**Software Quality & Security Tools:** Several studies emphasized tools that improve code quality and enforce better secure development practices. For example, Kushwaha et al. [6] implemented a DevSecOps pipeline incorporating Codacy, OWASP ZAP, and GitHub Advanced Security. This enables code quality improvements, helps detect vulnerabilities early, and makes regulatory compliance easier. On the other hand, Datar et al. [8] designed a custom tool for an insurance company that makes automated validations by implementing rule-based validations to ensure input correctness and improved validation accuracy.

Alternatively, Akin et al. [17] incorporate test automation tools such as JUnit and Selenium in a regression testing framework that the author designed and implemented in a banking application to decrease manual testing efforts and improve systems by pushing the organization to automated testing.

**Observability & Monitoring tools:** Monitoring tools were often employed to support real-time performance tracking and SLA compliance.

Mahida et al. [7] integrated these tools directly into DevOps pipelines to monitor system health, latency, and resource utilization. These tools helped the author to gain a faster feedback loop, support operational transparency objectives, and ensure compliance with the regulations. Panarin et al. [12] applied trace-based observability using a custom ClearTH framework to ensure transactional tracing and audit readiness. Schneider et al. [27] mentioned gains from using fuzz testing logs to uncover runtime anomalies of the mobile banking application.

However, several authors also uncovered the hurdles of implementing such tools into the legacy systems, which are common in the financial sector. According to some authors, the company's culture can also make it harder to implement it.

**Domain-Specific Frameworks and Tools:** Part of the studies used in this paper focused on frameworks specifically designed or tailored for the industry itself.

For example, Panarin et al. [12] present ClearTH, which is a domain-specific test automation framework used for the verification of transactions in the post-trade stage for investment banking. This test tool also helps with traceability. Hamad and Al Fatumi [11] describe the Scalable Quality and Testing Lab (SQTL), which focuses on high availability testing for mission-critical applications and helps with better test prioritization opportunities.

Even though these custom tools and frameworks are built with domain-specific requirements in mind, their implementation is not without hardship. Authors report challenges such as infrastructure limitations, cost, and the need for domain-expert configurations to work as intended.

### 4.3.2 Results of the Interviews

Interview responses highlighted three distinct categories in how professionals support software quality in financial projects:

1. **Tools** – These include CI/CD platforms (like Jenkins, GitLab CI, Azure DevOps), static analysis tools (SonarQube, SonarLint), scripting and orchestration tools, and custom dashboards. Tools were used to automate quality gates, enforce code policies, and manage pipelines across environments.
2. **Frameworks** – Participants mentioned widely used test frameworks such as JUnit, REST-assured, TestNG, and Spring Boot Test. Some also referenced deprecated or less-used frameworks like Selenium and SoapUI. Frameworks enabled standardized testing and were especially useful in backend validation and integration testing.
3. **Observability & Monitoring** – Tools such as ELK Stack (Elasticsearch, Logstash, Kibana), Prometheus, Grafana, and Splunk were commonly mentioned. These were used to monitor latency, track error rates, and observe system health during or after deployment. Observability was also linked to SLO/SLA enforcement and incident response.

These categories represent the technical foundation through which quality was maintained across development, testing, and operations.

Participant	Tools	Frameworks	Observability & Monitoring
P1	1	1	1
P2	1	1	1
P3	1	1	1
P4	0	0	0
P5	1	0	1
P6	1	1	1
P7	1	1	1
P8	0	0	0

P9	1	1	1
P10	1	1	1
Total	8	8	8

Table 10. Participants who referenced tools, frameworks, and observability practices in response to RQ3.

**Tools:** Eight out of ten participants discussed tools used to support CI/CD, static analysis, or orchestration in their quality workflows. These tools played a central role in enforcing process discipline, automating validations, and ensuring that the code met organizational standards before reaching production. Jenkins was the most commonly referenced tool, mentioned by multiple participants (P1, P3, P5, P6, P7, P10), often as part of automated pipelines. P3, a test engineer, explained how their entire quality flow runs through Jenkins, which is supported by internal dashboards. Similarly, P10, site reliability engineer, described their team's experience with static and dynamic code analyzers as *“SonarLint catches things in PR. We use it with Jenkins before builds. We also use SonarQube for metrics.”* Other tools included Bamboo (P1), GitLab CI (P1, P10), and internal SQL tools for data consistency checks (P5). These tools were embedded into daily developer workflows in many teams and tied directly to quality gates.

**Frameworks:** Eight participants referred to frameworks that enabled unit, integration, and API testing. These included widely used tools like JUnit, REST-assured, Spring Boot Test, TestNG, and SoapUI. For example, P6, tech lead, explained the importance of consistent frameworks for test portability: *“We use JUnit and REST-assured. The same framework across teams means coverage reports are standardized.”* By this statement, P6 also highlights the use of uniform tooling as a common practice. P7, senior software engineer in test, added that their team members *“use our internal regression suite based on REST-assured. It is tailored to backend APIs”*, reflecting how frameworks are used. Some frameworks, like Selenium, were mentioned as deprecated due to stability and flakiness (P6). However, their earlier usage showed how test frameworks evolved alongside system complexity.

**Observability & Monitoring:** Observability was mentioned across eight interviews, where participants described how their teams use dashboards, logging stacks, and real-time metrics to track quality after deployment. This included performance indicators, incident alerts, and SLA/SLO compliance. P9, a quant engineer, described using *“Grafana and Prometheus mostly. Kibana for deep logs. It’s critical for performance regressions,”* reflecting how visibility of system health supports their systems. P6, tech lead, reinforced the importance of monitoring for operational confidence, noting that they *“have dashboards for CPU, memory, error rates. It’s the*



*first thing we check after a release,”* showing how observability tools are used for quality assurance. Others described how these tools also informed pre-release decisions. P3, test engineer, mentioned performance dashboards were checked before allowing production release: *“Before going live, we verify the latency under load. We monitor for errors using dashboards.”*

These tools were used by operations and QA teams, confirming that observability has become a core component of modern quality assurance in financial systems.

Name	Description	Example
<b>Tools</b>	Tools used for CI/CD, static analysis, build orchestration, and automation across development pipelines. These include Jenkins, GitLab CI, Bamboo, SonarQube, SonarLint, and internal automation dashboards. Tools were embedded into quality gates and pre-release validation steps.	Jenkins was used to run full pipelines with code checks, test orchestration, and quality gating (P3, P7). SonarQube and SonarLint were integrated into PR reviews to prevent low-quality merges (P10). Some teams also used SQL-based internal tools to check data consistency and reporting (P5).
<b>Frameworks</b>	Standardized testing frameworks for implementing unit, integration, and regression tests. These ensured consistency in test structure and enabled API-level validation. Commonly mentioned frameworks include JUnit, REST-assured, TestNG, Spring Boot Test, and SoapUI.	REST-assured was used in several teams for API-level test cases (P6, P7). JUnit provided reusable test structures across teams, ensuring consistent results and coverage (P6). One team built a full internal regression suite using TestNG (P7).
<b>Observability &amp; Monitoring</b>	Tools and practices used to observe system behavior in production, verify SLO compliance, detect regressions, and manage incidents. This includes ELK stack (Elasticsearch, Logstash, Kibana), Prometheus, Grafana, Splunk, and custom dashboards.	Grafana and Prometheus were used for real-time metric visualization (P9). Kibana and ELK stacks support error rate and latency monitoring (P6, P10). Teams checked dashboards for performance spikes before and after releases (P3, P6).

*Table 11. Summary of tools, frameworks, and observability setups in the financial industry*



## 4.4 What are the advantages and disadvantages of implementing automated quality metrics in software development? (RQ4)

This subsection reports the results for RQ4 by analyzing the effects of automation on software quality. The results are focused on capturing the benefits and tradeoffs of automation. According to results, while automation improves the delivery speed, early defect detection, and consistency, it requires continuous investment and has its tradeoffs. The results show that automation can lead to flaky tests, false positives/negatives, and maintenance overhead.

### 4.4.1 Results of the SLR

Reviewed papers generally promote automation as a necessary enabler for speed, stability, and compliance in the context of quality within financial systems. However, its implementation discovered practical limitations tied to increased system complexity, team capabilities, and tooling.

**Benefits:** In most papers, automation is considered essential for improving quality. Several studies highlighted benefits such as early detection generation, manual effort reduction, and compliance enforcement as their primary gains.

For instance, Kushwaha et al. [6] implemented a DevSecOps pipeline with automated tests, dynamic and static code analysis tools, vulnerability detections, and security testing. Datar et al. [8] developed a BVCS platform for automating insurance rule validation, improving accuracy, and reducing manual QA effort by 80%. Akin et al. [17] applied Junit and Selenium for regression testing and made improvements by lowering manual effort and providing better consistency in testing outcomes. Xin et al. [29] even proposed a modular QTP-based automation framework that can be used in banking applications. This framework was designed to support layered tests and reuse the tests, also adding value to testing prioritization. The study reports that it reduced execution time by having parallelism, improved pass rates, and improved test process management. Another domain-specific framework author, Guo et al. [9] developed a FeT framework for a hybrid cloud mobile testing platform that uses real devices for parallel testing. The solution offers cost-efficient testing while maintaining compliance factors due to the sensitive information held by the banking app. Pardo et al. [10] emphasized in the paper that DevOps maturity is a key to success in automation, describing how automation features like blue/green testing and rollback mechanisms contribute to faster and more stable software delivery. Bucheger et al. [18], on the other hand, improved enterprise SOA testing by using a

component-based driver that enables automated orchestration opportunities and test resuability across the service layers. This approach helped in the early discovery of logic-level problems.

**Limitations:** Despite the benefits, automation can also be a limitation and mess up the test quality of the products. Problems such as flaky tests, false positives, tool immaturity, and team capability gaps were among the commonly cited limitations.

Challenges with testing, such as test stability and maintenance, were mentioned by Akin et al. [17]. He reported that not maintaining scripts caused failures due to system changes. Kushwaha et al [6] noted problems related to security scanners, particularly false positives, which drain development resources and decrease the trust in automated feedback. Hamad and Al Fatumi [11] discovered scalability and infrastructure problems when generalizing test automation in different environments. Xie et al. [14] mentioned that there is complexity in configuring a layered automation framework, specific problems such as test data management, driver scripts, and logic validators cause this complexity. Guo et al.[9] faced barriers while trying to integrate a cloud-based test solution to a legacy banking system, as well as barriers from regulatory compliance because of the sensitivity of the domain. Finally, Pardo et al. [10] underscored the need for DevOps culture and team maturity to achieve benefits from automation, as automation alone will not help with quality improvement if the team and organization are not supporting it.

#### 4.4.2 Results of the Interviews

RQ4 explores the advantages and disadvantages of automation. Few automation practices, such as test automation, CI/CD pipelines, and automated quality checks, were mentioned. Automation was perceived as a double-edged sword in most interviews: critical for maintaining speed and consistency, but expensive to implement and difficult to maintain under pressure. The benefits include faster feedback loops, higher test coverage, and reduced human error from manual testing. However, all the interviewers acknowledged some challenges with it, such as high initial cost, tech debt, test fragility, and significant effort. False positives/negatives, broken pipelines, and test maintenance were mentioned as factors that impacted delivery.

Participant	Automation Benefits	Automation Limitations	Automation Strategies
P1	1	1	1
P2	1	1	1
P3	1	1	1

P4	0	0	0
P5	1	0	1
P6	1	1	1
P7	1	1	1
P8	1	1	0
P9	1	1	0
P10	1	1	1
Total	9	9	7

Table 12. Automation Benefits, Limitations, and Strategies referenced by participants in response to RQ4.

**Automation Benefits:** Automation was praised by nine out of ten participants for enabling scalability, speed, and early feedback. It was commonly used in **test execution**, **build validation**, and **deployment readiness**. Several participants emphasized that automation directly contributes to releasing confidence and reduces the burden of manual testing. P1, a software engineer, highlighted that automation reduces overhead and enables faster delivery by stating, *“Automation removes overhead. Instead of manual verification, you focus on the core problem.”* Likewise, P6, tech lead, described the value of automation in regression testing and how it empowers fast iteration: *“Regression coverage is automated end to end. That’s what gives us speed. You don’t have to click around.”* Automation was also framed as essential in CI/CD pipelines. For example, P10, site reliability engineer, noted that automated testing is required to keep up with fast-paced environments by saying, *“if we didn’t automate quality checks, we’d delay every sprint. It keeps us moving.”*

In short, automation was not optional in most teams — it was embedded in their pipelines and considered key to modern delivery.

**Automation Limitations:** Although automation was widely adopted, nine participants discussed its drawbacks, particularly **maintenance complexity**, **flaky test suites**, and **false positives**. Several participants described situations where automated pipelines blocked releases for non-critical issues or failed silently. For example, P3, test engineer, mentioned that the test suite requires ongoing maintenance, which adds invisible cost: *“Sometimes tests fail for weird reasons. Then we spend hours debugging a pipeline that had nothing to do with our change.”* On

the other hand, P8, product owner, discussed problems with false positives leading to frustration: *“We get CI failures that aren’t reproducible locally. That’s hard to explain to stakeholders.”* Similarly, P2, a software engineer, pointed out that the unreliable tests can break the trust: *“if a test fails 30% of the time, people stop paying attention to it. Then it’s just noise.”*

The shared experience across teams was that **automation brings speed, but also cost**, particularly when systems are under-resourced or tests are not robust. These reflections suggest that automation without constant push can degrade the quality.

**Automation Strategies:** Seven participants discussed how their teams approached **automation planning, prioritization, and coverage**. Several described a focus on high-risk or high-impact paths, while others noted that automation efforts had to be constrained by time, team size, or tech maturity. P5, DevOps engineer, stated, *“We automate payments, reconciliation, anything with money. Frontend edge cases — not worth it,”* showing how automation is prioritized for business-critical functions. P7, senior software engineer in test, described their team's gradual approach: *“In the beginning, we just wrote tests. Later, we categorized them — critical paths first, then UI flows.”* It shows that automation had to evolve as systems grow in complexity. P1, a software engineer, emphasized pragmatism in choosing what to automate, acknowledging that *“you want 100% automation, but reality is time-bound. We write tests where failure is expensive.”*

These examples show that while automation was prioritized, most teams approached it **strategically and selectively**, often balancing business impact with available engineering effort.

Name	Description	Example
<b>Automation Benefits</b>	The advantages of automating tests and quality checks include reduced manual effort, faster feedback loops, and consistent release quality. Automation helps teams improve regression reliability and deliver code with higher confidence.	One participant explained that automation reduces overhead by eliminating manual checks, enabling teams to focus on problem-solving (P1). Others mentioned how full regression automation allows them to iterate quickly without rechecking everything manually (P6). P10 emphasized that sprint cycles would break without automation due to manual test bottlenecks.
<b>Automation Limitations</b>	The challenges associated with test automation include flaky tests, false positives, high maintenance costs, and broken trust in automated pipelines.	Participants described recurring issues like unreliable pipelines, false positives, and debugging time waste (P3, P8). P2 pointed out that teams often ignore failing tests if they lack consistency. These problems make

	These drawbacks can reduce developer confidence and introduce delivery friction.	test automation burdensome if not actively maintained.
<b>Automation Strategy</b>	The rationale behind deciding what to automate and where to invest effort is. Teams often first automate critical workflows, backend paths, or money-related systems while deprioritizing volatile or edge-case scenarios.	P5 shared that automation focused on financial systems, while UI edge cases were ignored. P7 described evolving their strategy by first covering critical APIs, then expanding to UI paths. Another participant noted that the automation scope is always constrained by time and risk prioritization (P1).

*Table 13. Summary of automation practices and trade-offs across financial teams*

## 4.5 What role do automated techniques like static and dynamic code analysis play in identifying vulnerabilities in complex codebases of the financial systems? (RQ5)

This sub-section reports the results for RQ5 by exploring the success of code analysis tools for vulnerability detection. The results show that these tools (static and dynamic analysis tools) positively affect the code quality and minimize code vulnerability and smells by integrating CI/CD pipelines. On the other hand, their effectiveness depends on the configurations and developer trust.

### 4.5.1 Results of the SLR

Given the sensitive nature of the financial industry, identifying vulnerabilities is key to protecting reputation and escaping financial loss in the industry. That is why many automated techniques are frequently used to improve threat detection, vulnerability scanning, and overall resilience. The studies show direct and indirect benefits of automation for identifying vulnerabilities. Even though automation is counted as having a positive effect, it still has some limitations in implementation, according to some papers.

**Automation for Security Scanning and Vulnerability Detection:** Kushwaha et al. [6] proposed an automated pipeline that integrates with static and dynamic code scanning tools (Codacity, OWASP ZAP, and GitHub Advanced Security), and by integrating them, the earlier issue identification was achieved compared to manual testing methods. The author reported several benefits, such as reduced security issues and improved development velocity, because of

the aforementioned automated testing techniques. Shneider et al. implemented fuzz testing to test mobile banking application security. This implementation allowed the system to discover numerous problems at runtime by injecting malformed inputs into the forms. The benefit was finding vulnerabilities that are easy to miss with manual testing; the paper's results highlight how a dynamic, randomized algorithm can help with testing. Xin et al. [29] proposed a custom, domain-specific automation framework. The authors reported some benefits, such as rapid re-testing of security, indirectly supporting vulnerability identification.

**Observability as Indirect Enablers of Vulnerability Detection:** The research discovered that observability can be an indirect enabler of Vulnerability Detection.

For example, Mahida et al. [7] highlighted the role of the observability tools (Prometheus, AppDynamics) in the pipelines. The author reports that these tools can be used for monitoring early warning mechanisms, anomaly detection, and resource usage. In a paper about the ClearTH, a test automation tool, Panarin et al. [12] demonstrated how traceable execution of post-trade banking transactions can be helpful for the detection of logical vulnerabilities and audit gaps.

**Implementation Scope and Limitations:** Even though most papers acknowledged the positive impacts of automation, some limitations were also discovered.

Guo et al. [9] described challenges integrating test environments with legacy systems, particularly regulatory restrictions that prevented full deployment of auto-scanning tools. On the other hand, Pardo et al. [10] mentioned that automation pipelines cannot be counted as an improvement if the organization is not mature enough for DevOps. Without proper support from the organization, the effectiveness of such tools in risk detection can be reduced.

## 4.5.2 Results of the Interviews

RQ5 explores how automated techniques are used to identify vulnerabilities in complex financial codebases and how development teams perceive their effectiveness. Participants discussed tools such as SonarQube, SonarLint, and internal rule engines integrated into CI pipelines. While these tools were widely adopted for enforcing coding standards and detecting common security flaws, interviewees also emphasized that the trust placed in these tools varied, depending on their accuracy and signal-to-noise ratio. A second recurring theme was the strategic use of these tools in gating pull requests and shaping quality decisions, where teams leveraged automated scanning as part of their CI/CD enforcement logic. Despite widespread usage, many participants noted limitations such as false positives, inconsistent severity scoring, and the need for manual oversight.

Participant	Automated Code Analysis	Enforcement Strategies
P1	1	1
P2	0	0
P3	1	0
P4	0	0
P5	1	1
P6	1	1
P7	1	1
P8	0	0
P9	0	0
P10	1	1
Total	6	5

Table 14. Categorized mentions under automated code analysis themes for RQ5

**Automated Code Analysis:** Six interviewees described their teams' use of static or dynamic code analysis tools, most commonly SonarQube and SonarLint, to detect bugs, code smells, and security vulnerabilities during development. These tools were typically integrated into pipelines or integrated development environments (IDEs) and used to scan every commit or pull request. However, participants frequently emphasized that these tools were only as valuable as their **configuration and output clarity**. For example, P1, a software engineer, explained that static analysis adds value only when developers trust what it reports by saying, *“Sonar is helpful when it’s configured well. Otherwise, you just get noise and people ignore it.”* P3, testing engineer, added that internal rule engines help external tools to reduce false positives and better reflect domain-specific concerns: *“We have our own internal rules layered on top of Sonar. Otherwise, it flags things that don’t matter in our stack.”*

Several participants, including P6, tech lead, and P7, senior software engineer in test, noted that these tools help surface hidden issues early but often trigger skepticism when results are inconsistent or lack business context. P6 remarked that *“sonar shows us the surface — but the real bugs need logic-level understanding.”*, while P7 stated that *“static analysis tells you where to look, not what’s wrong.”*



Overall, participants agreed that while automated analysis is helpful in **highlighting patterns**, its influence depends on developer trust, rule quality, and tuning to the organization’s actual risk profile.

**Enforcement strategies:** Five participants discussed how static and dynamic scanning tools are enforced through **pull request gates**, **pipeline blocks**, or **severity thresholds**. This often occurred through CI/CD systems like Jenkins or GitLab CI, which would prevent a merge or deployment if code did not meet quality thresholds. P5, DevOps engineer, explained that their team used severity filtering to determine when to allow deployment: *“High-severity issues block the pipeline. Medium or low, we review manually. It’s part of the release checklist.”* P10, site reliability engineer, described how analysis tools are used both as guidance and gatekeepers: *“Developers get SonarLint feedback in their IDEs. If it passes locally, it won’t block them. But the final merge still depends on the CI result.”*

However, some participants noted that overuse or strict policies could cause friction. P1 reflected on earlier experiences with overly aggressive PR gates: *“At some point, we had too many rules blocking PRs. It frustrated people. We turned it back.”* This statement shows how automation must be balanced.

Strategically, participants described **targeted enforcement**, focusing on **critical systems**, **high-risk modules**, or **shared codebases**, where consistency and code safety mattered most.

Name	Description	Example
<b>Automated Code Analysis</b>	Use of static code analysis tools (e.g., SonarQube, SonarLint) and internal rule engines to detect bugs, security issues, and code smells. While widely used, these tools' perceived reliability and impact vary based on configuration, accuracy, and developer trust.	P1 stated that static analysis becomes “just noise” if not tuned well. P3 mentioned layering internal rules on top of SonarQube to fit the organization’s stack better. P6 and P7 emphasized that these tools highlight surface-level issues but often miss deeper logic bugs.
<b>Enforcement strategies</b>	Strategies for integrating automated analysis into CI/CD workflows. Teams often enforce quality gates through PR checks or block pipelines based on issue severity. These practices aim to maintain baseline	P5 described blocking deployments based on issue severity levels, while P10 shared how SonarLint in the IDE and CI merge gates creates a multi-stage safety net. P1 reflected on the downside of too-strict



	quality while balancing speed and flexibility.	enforcement, which led to developer frustration.
--	--	--

*Table 15. Summary of automated code analysis practices for RQ5*

## 4.6 What impact does the prioritization of software quality have on feature release? (RQ6)

This sub-section reports the results for RQ6 and analyzes the effects of quality prioritization on delivery. According to results, quality prioritization tends to help with delivery speed and overall stability of the release process. However, it was discovered that there is a tradeoff between long-term stability and short-term delivery speed.

### 4.6.1 Results of the SLR

In the literature, multiple papers suggest that integrating automated tools and domain-related practices improves delivery speed and stability.

**Delivery Speed:** The Impact of embedding automated quality checks into the development pipeline contributed to faster delivery cycles.

Kushwaha et al. integrated static and dynamic scanning tools into the CI/CD pipeline. The author highlighted that this integration decreases bottlenecks of manual code review and contributes to faster delivery cycles. Also, it was mentioned that security test automation accelerated the release readiness by automating assurance. Guo et al. introduced the FeT hybrid cloud framework to enable parallel performance test execution, which helps to decrease the environmental setup time. As the paper reported, by integrating FeT, the team can validate releases faster. In addition, Xie et al. demonstrated a QTP-based regression framework, which, being modular, allowed rapid execution and re-execution of prioritized tests, cutting down QA time in the release and development stages, which in turn improved delivery speed. Pardo et al. described a DevOps maturity model that includes linked test automation, pipelines, and faster rollback readiness, making feature deployment streamlined.

**Delivery Stability:** While delivery speed is important and prioritized in the domain, stability is the primary return from sustained prioritization.

For instance, Panarin et al. [12] developed a ClearTH system in the paper specifically designed to trace the financial transactions and ensure the reproducibility of these transactions. The ability

to trace each transaction scenario added confidence to rollouts. However, Mahida et al. stated that observability tools incorporated into the pipelines made it possible to track real-time system insights, leading to safer deployments. The decreased rate of incidents, like SLA breaches, improved early intervention opportunities, thus stabilizing production deployments. Finally, Braun et al. highlighted that enabling expert evaluation of regression results via automated NLG reports decreased the likelihood of ambiguous shifts or missed risks in decision-making, thus enhancing controlled delivery practices.

#### 4.6.2 Results of the Interviews

Interviewees consistently acknowledged a trade-off between software quality and delivery speed. While beneficial in the long term, quality assurance practices were often cited as sources of short-term delivery delays. However, most participants emphasized that prioritizing quality leads to more stable systems, fewer incidents, and reduced rework over time. This theme emerged across roles, from software engineers to SREs and product owners, reflecting both technical and business-facing consequences.

To illustrate these patterns, Table X presents the distribution of responses across two main themes: (1) *Quality vs Speed Tradeoffs* and (2) *Impact on Delivery Timelines*.

Participant	Quality vs Delivery Speed Tradeoffs	Impact on Delivery Timelines
P1	1	1
P2	1	1
P3	0	1
P4	1	0
P5	1	1
P6	1	1
P7	1	0
P8	1	0
P9	1	1
P10	1	1

Total	9	7
-------	---	---

Table 16. Categorized mentions under Delivery Impact for RQ5

**Quality vs Delivery Speed Tradeoffs:** Most interviewees highlighted recurring tensions between business demands for rapid delivery and technical requirements for thorough quality validation. P4 stated, *“Stakeholders might want a new feature urgently, but we have to delay it until it meets security standards.”* Similarly, P7 shared that *“Business often wants rapid delivery and cost reduction, which can compromise quality,”* emphasizing pressure from tight timelines.

Some teams developed strategic release approaches to handle these tensions. P8 recalled, *“We once pushed a product on a Friday—it crashed, and we spent the whole evening fixing it. Midweek releases are much safer.”* This quote reflects the risk management tactics employed when time pressure threatens product quality.

Others emphasized balance. P6 mentioned aiming for a *“golden ratio: good enough quality to avoid major bugs but fast enough to ship.”* These insights indicate that quality prioritization is not binary but strategically calibrated.

**Impact on Delivery Timelines:** Respondents unanimously agreed that rigorous QA processes extend delivery timelines, particularly when involving security audits, compliance checks, or test coverage thresholds. P1 described how a penetration test led to immediate resource reallocation: *“We had to fix it, which delayed delivery—but it was essential for compliance and security.”* P10 reported that *“Every item must pass manual and automated testing. If it doesn’t, it’s delayed to the next release.”*

Still, many stressed the long-term benefits. P2 shared, *“Prioritizing quality increases release time in the short term but improves long-term stability and reduces bugs.”* P3 echoed this by saying, *“Skipping quality causes rework, which delays releases even more.”*

Name	Description	Example
<b>Quality vs Delivery Speed Tradeoffs</b>	The ongoing tension between business pressure for rapid delivery and engineering efforts to uphold quality standards. Teams often adapt by	P7 noted that “business often wants rapid delivery and cost reduction, which can compromise quality.” P8 described how pushing a product before readiness led to a crash and firefighting, emphasizing why they now avoid Friday releases. P6 spoke

	limiting risky releases, postponing features, or applying stricter validation to high-impact areas.	about aiming for a balance: “good enough quality to avoid major bugs but fast enough to ship.”
<b>Impact on Delivery Timelines</b>	Quality assurance activities such as test coverage enforcement, security scans, and manual validation processes can delay feature rollouts. However, teams view this as a strategic investment to prevent incidents and reduce technical debt in the long term.	P1 described how a penetration test delayed delivery, but stated it was “essential for compliance and security.” P2 acknowledged that “prioritizing quality increases release time in the short term but improves long-term stability.” P3 added that skipping QA often causes rework that delays future delivery even more.

*Table 17. Summary of Delivery Impact for RQ6*

## 5. Discussions

This section provides a discussion of the results in alignment with RQs. Section 5.1 aimed to discuss the results by analyzing each of the RQs; this section can be broken down into two paragraphs per research question. The first one integrates and summarizes the results, and the second paragraph discusses these results. Section 5.2 provides the framework designed and developed based on the results and discussion sections. It can be addressed as the product of the thesis, which will be a guide for the parties interested in the research. Finally, Section 5.3 outlines the study's limitations and suggests areas for improvement in future research.

### 5.1 Discussion of Research Questions

**(RQ1)** The first RQ addressed which software quality aspects are of priority for the financial systems. The results of both the SLR and the interview have some common prioritized aspects. Both of the methods mention aspects like security, performance, reliability, and compliance. Literature tends to emphasize security as a part of system requirements and as a foundational value. DevSecOps pipelines by Kushwaha et al. [6], fuzzing by Schneider et al.[27]. Moreover, several other papers, such as [9], [10], [29], discussed this aspect. Another aspect was audit-readiness, which we can consider as part of compliance. Papers like Mahida et al. [7] and Datar et al. [8] focused more on traceability enhancements by logging. Aspects such as reliability and performance are almost marked as the primary goals of any designed system in the domain. These were mentioned in a number of papers, such as [7], [10], [27]. Authors implemented systems to validate them via hybrid testing environments, big-data monitoring tools, or CI/CD pipelines. Interview results also repeated these findings. However, interviews can be considered to have more substantial alignment with practicality. Participants regularly named security and audit readiness (compliance) as non-negotiable. Some participants, such as P3 and P7, tend to state that reputational and financial risks drive these requirements. However, some interviews added a deeper context and knowledge to the security and audit readiness as arguments. For example, P6 mentioned compliance documentation, while candidates P2 and P10 shared that they use observability tools to ensure SLAs are met. This depth of context tends not to be covered in literature, or mentioned only indirectly. SLR results tend to frame maintainability or traceability as a secondary attribute. In comparison, interviews underplayed them by focusing more on live system behavior and operational readiness.

The suggestion can be made from the results that financial software quality is not abstract. It tends to be more enforceable and functional. Furthermore, it is more likely to be tied to business risk and external accountability. Compared to other aspects, a narrow focus on security and compliance might result from strict regulations and potential penalties. Also, the mission-critical

nature of the financial systems tends to show itself in a high focus on the performance and reliability of these systems. The less focus on developer experience or code maintainability can be understood as the internal engineering teams' priorities get deprioritized in favor of operational health. This issue could be a potential problem that can accumulate things such as technical debt and unmaintainable legacy systems in the long run. The root cause of the issue could be interpreted as the pressure of operational health and compliance obligations.

**(RQ2)** The second RQ aimed to explore how quality is measured in financial software systems. Both the SLR and interview results had several functional and non-functional metrics. However, we can say that interviews provided more granularity than the literature. The literature mentions functional metrics more in the context of tool usage and a high-level framework. For non-functional metrics, literature can be counted as linking them more to operational stability, compliance, and meeting system-level agreements. Literature has mentioned functional metrics as accuracy and test success rates in the context of tools like Codacity [6] or internal tools such as BVCS [8]. For non-functional requirements, metrics can be assumed to be more focused on system performance indicators like CPU and memory usage, latency, and throughput in the context of meeting system-level agreements (SLA/SLO). It can be stated that interviews had a more operational perspective by providing rich context, such as day-to-day use cases and examples. From participants' answers, it can be assumed that bug count, test coverage, and error rate are functional metrics that check the system's error proneness and audit readiness.. On the other hand, system performance metrics such as latency, uptime, throughput, and MTTR are often used and tracked by monitoring tools like Grafana, ELK, and Prometheus. These metrics can be agreed upon to provide helpful information for daily operations. A notable difference between interviews and literature that can be marked here is that six interviewers mentioned practical concerns in implementation, poor tooling, overlogging, and outdated dashboards (for example, P6, P10). This depth was covered thoroughly or not covered at all in the literature.

From the results, it can be suggested that both the literature and the interviews see the importance of the metrics. It can be seen that interviews have a more practical, hands-on approach, and the mentioned metrics are more fragmented compared to the literature, which seems to be more structured and theoretical. It can be implied that the concerns mentioned in interviews regarding misusing and misinterpreting the metrics underline the evolution processes of the long-lived financial systems. Seems like financial institutions are more focused on a few metrics, but having a complete picture of the system health.

**(RQ3)** The third RQ is aimed to examine what type of tools support the quality. The literature discusses a wide variety of tools and frameworks, including both general-purpose and domain-specific solutions. Solutions like Codacy, OWASP Zap, GitHub Advanced Security, and JUnit were the general-purpose ones, as well as some domain-specific ones like ClearTH and

SQTL. These tools can be grouped into several categories as they serve different purposes in the Software Lifecycle. We can see test automation tools, monitoring and observability tools, and some mentions of frameworks like DevSecOps in the literature. Authors tend to offer these integrations in order to improve audit readiness, CI/CD, built-in security scans, and automation regression tests. Interview participants tend to overlap the tools that are mentioned in the literature in their answers. Participants widely cited Jenkins, SonarQube, SonarLint, and REST-assured. Some also mentioned monitoring and observability tools, such as Grafana, ELK Stack, and Prometheus. The literature tends to focus more on architectural changes or scenario-specific changes. Interviews, on the other hand, tend to focus on developer experience and workflow integrations to get insights about real-time quality metrics and release management. For instance, P3 and P7 noted how the use of CI/CD tools and automated PR reviews can enforce quality gateways. P6 and P10 tend to describe how they use central quality dashboards to monitor the system and to track the releases. Both the literature and interviews tend to share information about the tools, but there are differences between their view of these tools. While interview results tend to review such tools in the context of organizational friction, business as usual solutions, learning curve barriers, and test flakiness and reliability, the literature tends to share details of how the solution can be implemented in a given scenario, and what kind of outcomes to expect from this integration. Also, literature tends to take infrastructure into consideration.

There is an overlap between the literature and interview results, which can be assumed to show that the financial sector is aligned with the tools that they use to ensure the quality. However, the implemented solutions seem to be highly context-dependent. The interviews tend to reveal that tooling is not just a technical concern; it is also embedded in the culture of the organization, team maturity, and risk appetite.

**(RQ4)** The fourth RQ aimed to investigate the role of automation in the software quality of financial systems. The literature praised automation as a critical component of financial software. Papers mentioned its benefits, such as early defect detection, security checks, reduced manual effort, and regression testing of code [6], [8], [17]. The literature mentioned the limitations in more detail in the context of tool complexity, challenges with infrastructure, and false positives and negatives [9], [29]. However, interviews can be assumed to address different aspects of it. Nine out of ten participants mentioned benefits from it, like faster feedback loops and automated integration (through CI/CD), as key benefits. Also, their answers tend to share negative context, such as flaky tests, maintenance costs, and trust issues with automated outputs. For example, participants like P3 and P8 mentioned that unreliable pipelines can cause issues with stakeholders and waste time with poorly configured tests. It can be seen that interviews added some missing details to the literature, like how it can limit engineering capacities and generate tech debt in the real world.



From the results, it can be interpreted that automation is not always beneficial. However, it depends more on what gets automated, how well it gets maintained, and who is responsible for maintaining it. Even though the literature tends to emphasize its potential, interviews added more details about the constraints that come with it. Constraints, like tech debt, increased system complexity, and maintenance effort. Another point that should be taken into consideration is how much automation is good enough—aiming for full automation or a more pragmatic approach by being selective through prioritizing it based on risk appetite. Literature tends to overlook this point by focusing more on the linear improvement. In interviews, a more realistic model is considered to be a dynamic investment that depends on the context.

**(RQ5)** The fifth RQ focused on the implementation practices of different software quality assurance capabilities. The literature tends to support automated solutions for detecting code vulnerabilities. Papers like Kushwaha et al.[6] integrate tools like Codacy, OWASP ZAP, and GitHub Advanced Security in their CI/CD pipelines. Schneider et al. [27], on the other hand, used fuzzing in their paper to expose code runtime issues in a bank's mobile app. Other papers like [9], [10], [14] discussed techniques like test reuse and cloud limitations that can influence the effectiveness of bug detection. From the interviews, it can be seen that a wide variety of tools, like SonarQube and SonarLint, are integrated into PR reviews and pipelines. On the other hand, interviews assumed more practical contexts than literature. These practical concerns were false positive/negative, noise-to-signal issue, and developer skepticism toward test accuracy (P1, P3, P6, P7, P5). Also, while literature tends to include observability in the vulnerability detection process, interviews tend not to include it in their view by using it for debugging rather than the detection of vulnerabilities.

Automated solutions can be functional in vulnerability detection; however, their effectiveness is believed to come with tuning and integrating them, and they can create trust issues. There is a divergence between the literature and the domain experts' view. The literature tends to overestimate tool efficiency in complex codebases by not accounting for the effort that it requires to maintain such solutions. Interviewees tend to have more skeptical thoughts about automated solutions, and even though they do not deny the benefits of having them, they also underscore the potential effort that is required to keep them working.

**(RQ6)** The sixth RQ is aimed to explore how compliance requirements affect quality assurance practices in financial systems. The literature tends to believe that prioritization of different software quality aspects helps to feature delivery speed and overall product stability. Some studies, such as those by Kushwaha et al. [6] and Guo et al. [9], report improvements in automation, which they believe help by reducing manual testing time. Others like Mahida et al. [7] and Panarin et al. [12] underscore the importance of observability and traceability in incident management. Interview participants tend to agree with the literature, and their answers offer a



more operational perspective. Most of these answers believe that there is a tradeoff of this kind of decision, like quality automation increases delivery time in the short term, even though it might make it stable in the long term. Also, it can be believed that adding automation can decrease time; however, it takes some effort to maintain it. (P2, P3, P5). Several participants also mentioned the issue with the pressure from the stakeholders that can push the team to release new features by pushing delivery timelines, which creates a need for negotiations with stakeholders (P4, P8, P9). There were examples of it, such as Friday Releases (P9) or audit gates that delay the release (P1, P10). These points tend to be overlooked in the literature. It can be assumed from a practical perspective that organizational processes can lead to delays, even though tools can increase the release speed.

The impact of quality prioritization can be assumed to be context-sensitive, which means it is not shaped by only the toolchain but also organizational processes and risk appetite. The literature tends to share a linear view compared to the interviews. While literature sees the result of automation as faster delivery and better results, interviews tend to underline that there is a need to take into consideration the negotiations, urgency, and strategies like midweek deployment and audit expectations.

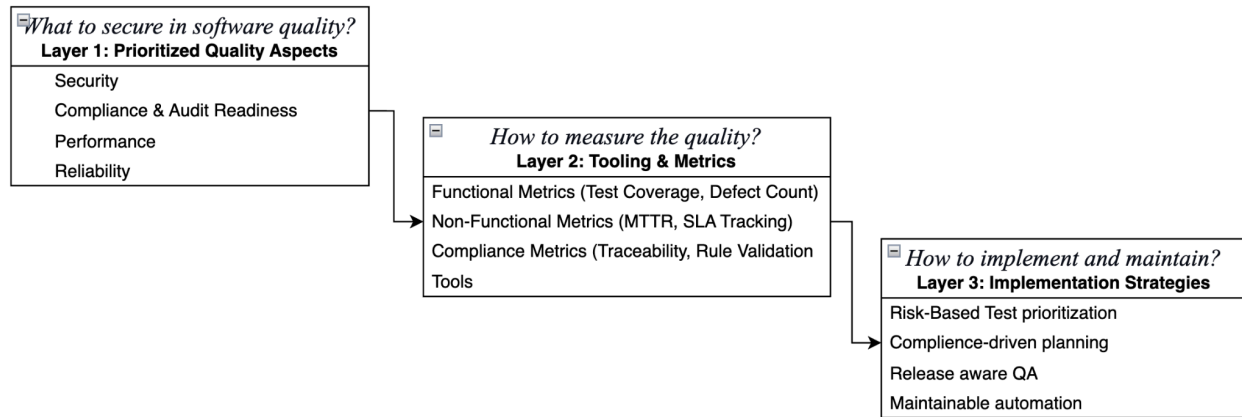
## 5.2 Framework

By synthesizing the results of the research, the following framework was developed. The framework is designed to be a practical guide on how software quality can be improved. The framework can be helpful for software engineers, QA specialists, project managers, and researchers who aim to explore the research topic further.

The framework is structured in three layers to answer the following questions:

- (1) What to secure in software quality?
- (2) How to measure the quality?
- (3) How to implement and maintain?

These layers are sequential and explore the topic from concept to execution. The graph representation of the framework is as follows in Figure 7.



*Figure 7. Framework Structure*

The integrated version of the framework can be found in Table 18. Each layer is discussed in more detail in Sections 5.1.1 through 5.1.3. The framework will be read from left to right. The first column, called Quality Aspects, captures the quality aspects that are prioritized by the industry. The second column, called Description, briefly explains the quality aspects. These two columns correspond to the results for RQ1. The third column, Metrics, aligns with RQ2 and captures how these aspects are measured. The fourth column, Tools&Frameworks, lists the tooling used to ensure quality; this column covers RQ3. The fifth column, the Implementation Practices column, aims to cover the implementation practices used in the particular aspect. The sixth column, Implementation Strategies, lists strategic solutions for the given aspect. The fifth and sixth columns cover results from RQ4 to RQ6.

Quality Aspect	Description	Metrics	Tools & Frameworks	Implementation Practice	Implementation Strategies	References
Security	Protection of the system from unauthorized changes, like sensitive data manipulations, exploits, and breaches	Bug Count, Defect Density, Test Success Rate, Model Accuracy.	OWASP ZAP, GitHub Advanced Security, DevSecOps pipelines, Codacy	automated testing, DevSecOps, static and dynamic code analysis, role-based access rules	Risk-based prioritization, Release-aware QA	Papers: [6], [27] Interviews: P3,P7, P8
Compliance & Audit Readiness	Systems should be audit-ready and comply with all regulatory requirements to protect the institutions.	Traceability, Trail completeness	ClearTH, SCTL framework, Jenkins, Splunk, Loki	logging for traceability, automated rule validations, and maintaining the existing system documentation	Compliance-driven planning	Papers: [8], [11], [12] Interviews: P7, P8
Performance	The system's performance measures whether the system can handle the workload in day-to-day and stressful situations.	Latency, Throughput, Response Time, SLA/SLO breaches	Elastic Stack (ELK), AppDynamics	load testing, performance monitoring, code optimizations, and analyzing past incidents.	Release aware QA	Papers: [7], [11], [14] Interviews: P3, P5, P9
Availability & Reliability	It measures whether the system is fault-tolerant and can deliver consistent service.	Uptime, MTTR	Prometheus, Grafana, Jenkins	health checks, rollback readiness, anomaly detection, tracking MTTR	Maintainable automation	Papers: [7], [28] Interviews: P2, P3, P6, P9

*Table 18. The framework*

### 5.2.1 Prioritized Quality Aspects

Like other high-stakes domains, the financial domain is under strict regulatory requirements and requires highly performant, reliable systems due to the sensitive nature of the data it handles. This fact was underlined both in the literature and in the interviews. The following table collects aspects that shape the quality assurance within the financial systems.

Prioritized Aspect	Reason	Common Practices	References
Security	Financial systems have security as a primary quality requirement to protect user data.	Implementation of automated testing in CI/CD pipelines. DevSecOps pipelines, static and dynamic code analysis, role-based access rules, and encryption are common practices to address security risks.	[6], [27]
Compliance	Strict regulatory requirements push the financial products to maintain audit readiness, traceability, and reproducibility of the operations.	Designing and implementing logging for traceability, automated rule validations, and maintaining the existing system documentation while changing features are common practices to support compliance.	[8], [11], [12]
Performance	The financial systems need to support real-time transactions, which require the system to work and be performant under high load.	Load testing, monitoring latency and throughput with monitoring tools and monitoring system SLA/SLOs, and analyzing existing incidents are common practices to address the performance issue.	[7], [14], [28]
Reliability	Failure to complete operations and downtime of such systems can cause disruption and increase the risk of financial and	Monitoring uptime and system health, implementing better traceability through logs, tracking MTTR (mean time to recovery), implementing central dashboards, and resource redundancy can help to address the reliability issue.	[7], [28]

	reputational damage. These risks make system stability essential.	Failure to complete operations and downtime of such systems can cause disruption and increase the risk of financial and reputational damage. These risks make system stability essential.	
--	---	---	--

*Table 19. Prioritized aspects of quality in financial systems*

### 5.2.2 Metrics and Tooling

Several functional and non-functional metrics are used to measure the quality of the prioritized aspects. Given that several compliance-related metrics were discussed in the literature and interviews, we decided to separate them in the framework, even though they are technically part of non-functional metrics. The metrics are collected using automated tests, observability and monitoring tools, and reporting systems.

Type	Metrics	Tools	Context	References
Functional	Bug count, defect density, test success rate, and model accuracy.	SonarQube, JUnit, REST-assured, TestNG, internal dashboards, Jenkins (CI/CD) pipelines.	These metrics assess the overall code quality and validate the functional correctness of the system.	[6], [14], [17]
Non-functional	Uptime, latency, throughput, MTTR, error rate, and resource usage.	Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana), Splunk.	These metrics are used to evaluate system behavior in day-to-day operations and detect incidents and breaches.	[7], [28]
Compliance Related	Traceability, rule validation coverage, and structured test documentation.	Custom platforms like BVCS, Jenkins, GitLab CI, structured loggers, and	These metrics support internal and external audit processes by implementing validation reports and	[8], [11], [12]

		internal reporting tools.	logging.	
--	--	---------------------------	----------	--

Table 20. Metrics are used to measure software quality.

### 5.2.3 Implementation Strategies

While the questions regarding aspects, metrics, and tooling are answered in the sections above, this section will discuss the implementation strategies. Teams that develop and maintain the financial systems tend to use various implementation strategies; each has its own benefits, challenges, and tradeoffs.

Strategy name	Best Practices	Benefits	Challenges and Trade-offs	References
Risk-based prioritization	Focusing on test automations for business-critical system functionality. For example, business flows like trading, payment processing, and regulatory reporting.	Makes the high-impact feature delivery smooth and increases team confidence. Also, reduces over-testing in low-risk functionality.	Can result in overlooking bugs in less business-critical functions by under-testing such functions.	[8], [27]
Compliance-driven planning	Including and prioritizing regulatory requirements in the early phases of planning.	Helps to avoid last-minute audit issues or fines.	Can be a constraint for design flexibility and system architecture. Can add an extra layer of approvals for delivering the feature, which decreases the delivery speed.	[8], [11], [12]
Release aware QA	Use internal quality gates aligned with prior development	Improves delivery and overall system	Can be a constraint to urgent releases by adding an extra layer	[6], [7], [10]

	experience. The quality gates can be implemented by using CI/CD pipelines and automated PR reviews.	stability.	of approvals.	
Maintainable automation	Investing in small modular test suites that will be easy to support and maintain in the future. Monitoring and fixing test flakiness as it occurs. Enforce ownership for automated testing solutions to support their maintenance.	Helps to reduce manual testing. Improves delivery speed by implementing atomic regression tests	High initial investment. Also, requires team discipline to add and maintain the tests	[12], [17], [27]

*Table 21. Implementation strategies*

## 5.3 Limitations

The research faces limitations inherent to its methodologies. Both Systematic Literature Review (SLR) and qualitative interview methodologies have their constraints. For SLR methodology, there are two primary limitations:

- (1) The risk of missing relevant studies despite having a structural search procedure.
- (2) The risk of not extracting all the relevant data from the referenced literature.

For the Interview methodology, two primary limitations are as follows:

- (1) Limited generalization capability, given the small sample size of participants.
- (2) Internal validity of extracted data, such as bias and subjective interpretation risk, during the data analysis phase.

The first SLR limitation, the risk of missing relevant studies, is described in guidelines by Kitchenham [19]. The research mentioned that even carefully constructed search strings and inclusion/exclusion criteria can miss relevant studies, as the discovered literature is limited to

databases and keyword choice. This thesis contains papers chosen by publication date, publication type, domain alignment, and language. These criteria could lead to missed relevant studies. To address this problem, the search string was iteratively tested. The second limitation is the risk of not extracting all the relevant points from the literature, which is also mentioned by Breteton et al.[21]. It is mentioned that important findings can be missed if reported unclearly. In the paper, this issue is addressed by using the extraction form and thematic analysis. Despite the mentioned countermeasures, the risks are not entirely eliminated.

The first interview limitation, limited generalization capability, is due to the sample size of the interviews. The research presents results from ten interview participants, a small sample size, to present broad industry insights. To address this issue, the participants were chosen from diverse backgrounds. The role variety described in Table 4 improves the coverage of different viewpoints and reduces the bias. The other risk was subjective interpretation, as qualitative research depends on the researcher, and the interpretation bias can be overlooked. To address this issue, the refined and anonymized transcripts were reviewed, and coding for thematic analysis was conducted in three rounds. Despite the mentioned countermeasures, the risks are not entirely eliminated.



## 6. Conclusion

This thesis aimed to explore how software quality is planned and implemented in financial software systems. The research was conducted by using a combination of systematic literature reviews (SLR) and qualitative interviews. The SLR results combine the 14 domain-relevant publications that met defined inclusion and exclusion criteria. These publications reveal an emphasis on security, reliability, performance, and regulatory compliance. Interviews with 10 domain experts were conducted to explore topics in depth. The interview results complemented the thesis by providing practical insights into automation, tooling, and metrics within the domain's real, day-to-day, business-as-usual scenarios.

The key delivery of this thesis is a structured framework that integrates the findings of pre-defined research questions (RQs). The framework can be a minimal guide to help define, prioritize, and implement software quality in such systems. This framework was designed with the goal of supporting practitioners and researchers by providing insights from both existing literature and a practical view from domain experts. Additionally, a discussion section can be helpful for more details, as it was developed to interpret and compare the SLR and interview results and highlight key alignments

### 6.1 Future Work

Several perspectives for future work were identified from the research itself and its results. First, the systematic literature review can be expanded by providing more keywords, which can be done by adding alternative terminology for software quality and automation, and a search string to capture additional relevant studies. This could expand the scope of the SLR results further. Second, the interview study can be extended by increasing the number of domain experts and adding more diverse roles. Since this thesis included perspectives from experts with technical and managerial experience, future work could include auditors, compliance officers, and business risk managers to provide more depth to the topic by diversifying the perspectives. Finally, the proposed framework can be refined and validated through practical implementation. Future work can help by conducting case studies in the companies of the domain or practical implementation in real-world projects. Applying the framework in real-world projects and analyzing the results by tracking proposed solutions of this paper, such as impact on delivery, audit readiness, automation bottlenecks, and quality metrics, would allow researchers to assess the current frameworks' practicality and make improvements based on the observed outcomes of the implementation.

## References

- [1] V. Ratten, “Technological innovations in the m-commerce industry: A conceptual model of WAP banking intentions,” *J. High Technol. Manag. Res.*, vol. 18, no. 2, pp. 111–117, Jan. 2008, doi: 10.1016/j.hitech.2007.12.007.
- [2] C. Nagy, “Static Analysis of Data-Intensive Applications,” in *2013 17th European Conference on Software Maintenance and Reengineering*, Genova: IEEE, Mar. 2013, pp. 435–438. doi: 10.1109/CSMR.2013.66.
- [3] O. O. Top, B. Ozkan, M. Nabi, and O. Demirors, “Internal and External Software Benchmark Repository Utilization for Effort Estimation,” in *2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, Nara, Japan: IEEE, Nov. 2011, pp. 302–307. doi: 10.1109/IWSM-MENSURA.2011.41.
- [4] A.-J. Molnar, A. Neamțu, and S. Motogna, “Evaluation of Software Product Quality Metrics,” in *Evaluation of Novel Approaches to Software Engineering*, vol. 1172, E. Damiani, G. Spanoudakis, and L. A. Maciaszek, Eds., in Communications in Computer and Information Science, vol. 1172. , Cham: Springer International Publishing, 2020, pp. 163–187. doi: 10.1007/978-3-030-40223-5\_8.
- [5] K. A. Aziz, S. Abdullah, M. A. Jabar, R. N. H. Nor, and Y. Y. Jusoh, “Internet Banking Acceptance Towards Improving Sustainability: A Systematic Literature Review,” in *2023 International Conference on Information Management (ICIM)*, Oxford, United Kingdom: IEEE, Mar. 2023, pp. 56–60. doi: 10.1109/ICIM58774.2023.00016.
- [6] M. K. Kushwaha, P. David, and G. Suseela, “Automation and DevSecOps: Streamlining Security Measures in Financial System,” in *2024 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, July 2024, pp. 1–6. doi: 10.1109/CONECCT62155.2024.10677271.
- [7] A. Mahida, “Integrating Observability with DevOps Practices in Financial Services Technologies: A Study on Enhancing Software Development and Operational Resilience,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 15, no. 7, 2024, doi: 10.14569/IJACSA.2024.0150701.
- [8] A. Datar, A. Zare, A. A. R. Venkatesh, S. Kumar, and U. Shrotri, “Automated Validation of Insurance Applications against Calculation Specifications,” Sept. 08, 2022, *arXiv*: arXiv:2209.03558. doi: 10.48550/arXiv.2209.03558.
- [9] C. Guo, S. Zhu, T. Wang, and H. Wang, “FeT: Hybrid Cloud-Based Mobile Bank Application Testing,” in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Lisbon: IEEE, July 2018, pp. 21–26. doi: 10.1109/QRS-C.2018.00018.
- [10] C. Pardo, J. Guerrero, and E. Suescún, “DevOps model in practice: Applying a novel reference model to support and encourage the adoption of DevOps in a software development company as case study,” *Period. Eng. Nat. Sci.*, vol. 10, no. 3, pp. 221–235, 2022, doi: 10.21533/pen.v10i3.3086.
- [11] R. M. H. Hamad and M. Al Fayoumi, “Scalable Quality and Testing Lab (SQTL): Mission-Critical Applications Testing,” in *2019 International Conference on Computer and Information Sciences (ICCIS)*, Sakaka, Saudi Arabia: IEEE, Apr. 2019, pp. 1–7. doi: 10.1109/ICCISci.2019.8716404.

- [12] V. Panarin *et al.*, “Poster: ClearTH Test Automation Framework: A Running Example of a DLT-Based Post-Trade System,” in *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, Apr. 2019, pp. 358–362. doi: 10.1109/ICST.2019.00042.
- [13] D. Braun, A. Sajwan, and F. Matthes, “User-adaptable Natural Language Generation for Regression Testing within the Finance Domain:,” in *Proceedings of the 22nd International Conference on Enterprise Information Systems*, Prague, Czech Republic: SCITEPRESS - Science and Technology Publications, 2020, pp. 613–618. doi: 10.5220/0009563306130618.
- [14] X. Xie, Z. Yang, J. Yu, and W. Zhang, “Design and implementation of bank financial business automation testing framework based on QTP,” in *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*, Changchun: IEEE, Dec. 2016, pp. 143–147. doi: 10.1109/ICCSNT.2016.8070136.
- [15] P. Ciancarini, A. G. Nuzzolese, V. Presutti, and D. Russo, “SQuAP-Ont: an Ontology of Software Quality Relational Factors from Financial Systems,” *Semantic Web*, vol. 11, no. 6, pp. 1007–1021, Oct. 2020, doi: 10.3233/SW-200372.
- [16] M. Sirshar, Z. Alam, G. Ejaz, and M. Mumtaz, “Software Quality Assurance in Safety Critical Systems,” Dec. 10, 2019. doi: 10.20944/preprints201912.0130.v1.
- [17] A. Akin, S. Senturk, and V. Garousi, “Transitioning from Manual to Automated Software Regression Testing: Experience from the Banking Domain,” in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, Nara, Japan: IEEE, Dec. 2018, pp. 591–597. doi: 10.1109/APSEC.2018.00074.
- [18] G. Buchgeher, C. Klammer, W. Heider, M. Schüetz, and H. Huber, “Improving Testing in an Enterprise SOA with an Architecture-Based Approach,” in *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Apr. 2016, pp. 231–240. doi: 10.1109/WICSA.2016.24.
- [19] B. Kitchenham, “Guidelines for performing Systematic Literature Reviews in Software Engineering,” vol. 2, 2007.
- [20] O. A. Adeoye-Olatunde and N. L. Olenik, “Research and scholarly methods: Semi-structured interviews,” *JACCP J. Am. Coll. Clin. Pharm.*, vol. 4, no. 10, pp. 1358–1367, Oct. 2021, doi: 10.1002/jac5.1441.
- [21] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, “Lessons from applying the systematic literature review process within the software engineering domain,” *J. Syst. Softw.*, vol. 80, no. 4, pp. 571–583, Apr. 2007, doi: 10.1016/j.jss.2006.07.009.
- [22] Y. Levy and T. J. Ellis, “A Systems Approach to Conduct an Effective Literature Review in Support of Information Systems Research,” *Informing Sci. J.*, vol. 9, pp. 181–212, 2006, doi: 10.28945/479.
- [23] C. Okoli, “A Guide to Conducting a Standalone Systematic Literature Review,” *Commun. Assoc. Inf. Syst.*, vol. 37, 2017.
- [24] A. Fink, *Conducting Research Literature Reviews: From the Internet to Paper*. Thousand Oaks, California: Sage Publications, 2014.
- [25] M. I. Alhojailan, “Thematic analysis: A critical review of its process and evaluation,” *West East J. Soc. Sci.*, vol. 1, no. 1, pp. 39–47, 2012.
- [26] Gareth Terry, Virginia Braun, and Victoria Clarke, “Thematic Analysis,” in *The SAGE*

- Handbook of Qualitative Research in Psychology*, 2nd ed., London, 2017, pp. 17–37.
- [27] M. A. Schneider, M.-F. Wendland, A. Akin, and S. Sentürk, “Fuzzing of Mobile Application in the Banking Domain: a Case Study,” in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Dec. 2020, pp. 485–491. doi: 10.1109/QRS-C51114.2020.00087.
- [28] K. Sathupadi, S. Achar, S. V. Bhaskaran, N. Faruqui, and J. Uddin, “BankNet: Real-Time Big Data Analytics for Secure Internet Banking,” *Big Data Cogn. Comput.*, vol. 9, no. 2, p. 24, Jan. 2025, doi: 10.3390/bdcc9020024.
- [29] B. Xu, M. Chen, C. Liu, J. Li, Q. Zhu, and A. J. Kavs, “Quantitative Quality Evaluation and Improvement in Incremental Financial Software Development,” in *Computer, Informatics, Cybernetics and Applications*, vol. 107, X. He, E. Hua, Y. Lin, and X. Liu, Eds., in *Lecture Notes in Electrical Engineering*, vol. 107. , Dordrecht: Springer Netherlands, 2012, pp. 1453–1461. doi: 10.1007/978-94-007-1839-5\_156.

# Appendix

## I. Interview Guideline

### Research Questions:

RQ1: What critical aspects of software quality are generally prioritized by the financial industry?

RQ2: What functional and non-functional software quality metrics are defined and used in different financial products?

RQ3: What methodologies and tools are most effective for measuring and assessing software quality?

RQ4: What are the key advantages and disadvantages of implementing automated quality metrics in software development?

RQ5: How influential are automated techniques like static code analysis in identifying vulnerabilities in complex codebases of the financial industry?

RQ6: What impact does the prioritization of software quality have on future rolling?

**Materials, software, and environment:** pdf of this document, pdf of interview questions, consent form, A4 notebook for answers, a pen, a laptop, a charger, headphones, Zoom/Google Meet calls.

**Introduction:** The University of Tartu's research group conducted the study. The study's objective is to research how the data quality of the software is implemented in the Financial sector. The study aims to develop a framework from existing research and industry experts. The study is part of the Master's thesis of a student at the University of Tartu.

### Procedure:

Pre-interview:

1. Send the consent form to the interview participants and ask them to sign it before the interview process. Upon receiving it, assign an index number to the file and store it in a separate designated folder.
2. Send a Zoom or Google Meet link to the participant.
3. Create a calendar (Google Calendar) event for the interview and invite the participant. Add the interview link to the event description.
4. Check if all the abovementioned materials before the interview starts.

#### Interview:

1. Welcome to the participant.
2. Introduce yourself.
3. Thank them for their participation.
4. Introduce the study.
5. Start video recording.
6. In case the participant did not send the signed consent form, walk them through the consent form and ask for their verbal consent on the record.
7. Conduct the interview.
8. Ask extra questions for additions and clarifications.
9. Stop recording.
10. Thank them for participation.

#### Post-interview:

1. Convert the video recording into the transcript.
2. Recheck the interview transcript.
3. Assign an index number to the transcript and store it in a designated folder separate from the consent folder.
4. Mark the initial findings from the interview.

## II. Consent Form

**Research title:** Securing Quality in Software Development Process

**Research student:** Kamil Aliyev, Master's Student at the University of Tartu

**Supervisor:** Frederick Milani, Associate Professor of Information Systems, University of Tartu

**Introduction:** This study is conducted by a research group from the University of Tartu (Estonia), and is part of a research student's Master's Thesis. The primary research objective of this thesis is to identify how software quality is being implemented in modern financial products. The study aims to develop a framework for securing data quality in the financial industry. The framework will be an organized system that helps software engineers, project managers, and business analysts. The main research questions are targeted to identify critical aspects of software quality prioritization, functional and non-functional metrics, and tools used for it.

During the study, the researcher will conduct an interview to discuss the topics mentioned above. The researcher will record the conversation during the study.

**Participation requirements:** In order to participate in the interview, a person should:

- 1) be at least 18 or older
- 2) have a work experience in the financial sector at least one year
- 3) have experience with implementing software quality
- 4) be fluent in English

**Expected interview duration:** The interview process will take approximately 1 hour

### **Risks and Benefits:**

**Privacy and Confidentiality:** In order to protect the interviewee's privacy, the research team will follow the following procedure. The original recordings will only be accessible to the Researcher and Supervisor. The audio recording of the interview will be transcribed, potential identifiers will be removed or aggregated, and the original recordings will be deleted afterward. Your data and consent form will be kept separate. Your consent form will be stored securely and will not be disclosed to any third parties.

By participating, you agree that the inputs that you provide and the information that was gathered during this interview will be used by the researcher and the university for publication purposes. The University of Tartu requires all research data to be stored for at least five years following the day when this report will be published. Thus, the researcher will store all the provided

information, except the data that can identify you, for the abovementioned timespan.

**In the case of any inquiry about the research:** If you have any comments, concerns, or other inquiries about the study, whether it is pre- or post-interview, please contact the Researcher (Kamil Aliyev, kamilali@ut.ee)

**Voluntary Participation:** Your participation in this study is voluntary. You may discontinue participation at any time during the research activity.

I verify that I am 18 or older. I have read and understood all the information provided above, and I want to participate in this study:

a) Yes    b) No

**Participant:** As a participant, I have read the provided information, and all my questions about the document have been answered. I understand that as this study progresses, I am welcome to voice any issues, questions, or complaints I may have regarding any component of the study. I acknowledge that the researcher, the supervisor, or any other study participant has addressed my feedback.

**Researcher:** I state that I have fully informed the participant about the study, its purpose, and the goals of the research being conducted. I briefed the participant about the potential benefits and possible risks related to the involvement in the study. We have addressed all of the participants' inquiries regarding the study, and we remain accessible to handle any other inquiries that may come up in the future.

**Signature:**

Participant:                      (placeholder)

Researcher:                      (placeholder)



### **III. Interview Questions**

Hello! How are you? It is a pleasure to meet you.

Thanks for investing your time and volunteering for the study!

As I mentioned during our first communication, I am researching Securing Quality in the Software Development Process of Financial Industries. Now, I would like to start recording, then, we can go through the Consent Form, which will take around 3-4 minutes, and ask for your verbal consent to start the interview. Is this plan okay for you? If you have any questions regarding the interview, please feel free to ask them now.

#### **Interview Questions:**

1. How is the software quality of the projects planned in the early design and planning phase?
  - 1.1. What priority does software quality have in project planning and requirements gathering?
  - 1.2. How do you define the quality goals of the project?
  - 1.3. Do you use any specific methodologies or frameworks from the beginning of the project?
2. What functional and non-functional quality metrics are generally used by your team?
  - 2.1. What challenges do you face while implementing these metrics to the project?
  - 2.2. How do these metrics differ from project to project?
3. What critical aspects of software quality are generally prioritized by your company?
  - 3.1. What effects do these priorities have on service level objectives (SLO)?
  - 3.2. Did you come across with software quality aspects that conflict with business objectives?
4. What frameworks, tools, and methodologies are most common and vital for projects in the financial industry?
  - 4.1. Have there been any tools or methodologies that could have met your expectations?
5. What are the key advantages and disadvantages of implementing automated quality metrics in software development?
  - 5.1. Can you discuss any concerns or limitations you have encountered with automation in quality metrics?
6. What impact does the prioritization of software quality have on future rolling?
  - 6.1. How has focusing on quality affected productivity and project delivery time?

- 6.2. Can you describe the decision-making process behind prioritizing quality over speed or new features?
- 7. How does the organization ensure compliance with local and global regulations related to the software quality of the financial projects?
  - 7.1. Can you describe your challenges in maintaining compliance during software development and updates?

#### IV. Final Tag List with Definitions for Interviews

Final Themes	Related RQ(s)	Definitions
Key Quality Priorities	RQ1	Understand the critical quality prioritization in finance due to high-risk nature and the need for stability.
Conflicts Between Quality and Business Demands	RQ1, RQ6	Explore tensions between delivering fast and maintaining high QA standards in financial software.
Functional & Non-Functional Quality Metrics	RQ2	Identify how financial products use functional and non-functional metrics to measure quality.
Metric Implementation Challenges	RQ2	Examine obstacles in implementing metrics across varied products and systems.
QA Tools, Frameworks & CI/CD	RQ3	Identify effective tools and methods for assessing software quality in real-world financial workflows.
Monitoring & Observability Practices	RQ3	Understand how system performance and errors are tracked.
Automation Practices & Strategy	RQ4	Analyze how automated QA is planned, prioritized, and integrated into financial software development.
Benefits & Limitations of Automation (incl. Static Analysis)	RQ4, RQ5	Explore the effectiveness, risks, and limitations of automation techniques (e.g., static code analysis) for quality assurance.
Quality vs Speed Tradeoffs in Delivery Planning	RQ6	Understand how organizations make decisions between speed and quality in planning rollouts and features.
Impact of QA on Delivery Timelines	RQ6	Assess the measurable effects of QA prioritization on delivery speed, rollout reliability, and incident reduction.

# License

## Non-exclusive licence to reproduce the thesis and make the thesis public

I, Kamil Aliyev,

1. Herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,  
**Securing Quality in the Software Development Process of Financial Systems**  
Supervised by Fredrik Payman Milani.
2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified on p. 1 and 2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Kamil Aliyev  
01/08/2025