CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Relationship between flaky tests and requirements

Master's Thesis in Computer science and engineering

Iqra Ammad

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's Thesis 2025

Relationship between flaky tests and requirements

(Iqra Ammad)

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Relationship Between Flaky Tests and Requirements
Iqra Ammad

Supervisor: Irum Inayat, Department of Computer Science and Engineering
Advisor: Abdul Rauf, Test Scouts AB
Examiner: Jennifer Horkoff, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Iqra Ammad
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Flaky tests produce inconsistent results without code changes, representing a significant challenge in software development by undermining developer confidence and delaying continuous integration processes [33]. Despite their recognized impact on software quality assurance, the relationship between requirements quality and test instability remains underexplored in industrial contexts. This study investigates how industry practitioners perceive the relationship between requirements quality and flaky tests and identifies specific requirements quality attributes that most significantly contribute to this problem, providing evidence-based guidelines to mitigate these issues. A mixed-methods approach was employed by combining quantitative survey data from 74 industrial practitioners with qualitative semi-structured interviews from eight professionals across diverse domains, including automotive, banking, healthcare, and software development. The findings establish flaky tests as a widespread phenomenon, with 80% of practitioners reporting regular encounters with test flakiness in their projects. The study reveals a notable awareness practice gap: while 80% of practitioners recognize that requirement completeness frequently impacts test stability, only 56% consistently consult requirements during flaky test investigations. This selective behavior reflects sophisticated contextual reasoning rather than a lack of knowledge, with practitioners strategically assessing when requirements-related factors are likely contributors versus when technical diagnostics are more appropriate. The critical requirements quality attributes emerge as contributors to test flakiness: completeness deficiencies (including unspecified timeout durations affecting 77% of practitioners, missing acceptance criteria at 61%, and incomplete data specifications at 62%), and ambiguity issues (especially unclear success and failure conditions reported by 65%). The research identifies timing specifications as a critical blind spot in current requirements practices, often treated as implementation details rather than explicit requirements, leading to cascading effects where independent timing assumptions by developers and testers create environment sensitive test behavior. The study demonstrates significant domain-specific variation in the requirements-flakiness relationship, with practitioners reporting varying levels of requirements related contributions across different sectors. Structured domains with formal processes (banking, healthcare) show lower requirements related flakiness, while less regulated environments experience higher impacts. This contextual dependency reflects how organizational processes, regulatory constraints, and system criticality mediate the impact of requirements quality on test stability. Based on empirical findings, the research presents evidence-based guidelines organized around fundamental requirements quality attributes to systematically reduce requirements related test flakiness. These guidelines provide systematic approaches for organizations to reduce requirements-related test flakiness through targeted interventions addressing the identified quality gaps. The study establishes requirements quality as a measurable contributor to software system reliability, with practitioners estimat-

ing that 21-60% of flaky tests originate from requirements related issues, deserving systematic attention alongside traditional technical factors in software quality assurance efforts.

Keywords: Flaky tests, Requirements quality, Test stability, Completeness, Ambiguity, Software development

Acknowledgements

I would like to sincerely thank Irum Inayat, my supervisor at Chalmers, for her encouragement, support, and honest feedback. I am also grateful to Jennifer Horkoff, my examiner, for her valuable input, understanding, and guidance throughout the research process. I extend my sincere thanks to all the practitioners who generously took the time to speak with me; your insights and reflections made this work possible. I am also thankful to my industrial supervisor, Abdul Rauf, at Test Scouts AB.
To my children, thank you for your patience, understanding, and for always being there for me. To my mother and sisters in my home country, thank you for always believing in me and encouraging me to do my best. I truly could not have done it without all of you. To my husband, for sharing insights and advice from his own academic journey.
Finally, to my friends, especially Sobia, without her belief, constant encouragement, and support, this would not have been possible. Thank you for cheering me on, listening when I needed you, and reminding me that I could do this. Lastly, I would like to thank my parents-in-law for their support throughout my thesis journey.

Iqra Ammad, Gothenburg, October 2025

Contents

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**
1.1 Problem Description 1
1.2 Purpose of the Study 2
1.3 Significance of the Study 3

2 Background and Related Work **5**
2.1 Requirements Engineering and Quality 5
2.2 Requirements Quality Attributes 6
2.3 Software Testing Fundamentals 7
2.4 Flaky Tests 8
2.5 Requirement Engineering and Software Testing 10
2.5.1 Limited Research on Upstream Flaky Tests Causes 11
2.6 Summary 11

3 Methods **12**
3.1 Research Design : An Exploratory Case Study 12
3.2 Data Collection 13
3.2.1 Quantitative Study: Survey 13
3.2.2 Qualitative Study: Semi-Structured Interviews 14
3.3 Data Analysis 16
3.3.1 Quantitative Analysis 16
3.3.2 Thematic Analysis 17

4 Results **19**
4.1 Survey Findings 19
4.2 Thematic Analysis of Interview Data 24
4.2.1 Overview of Codes and Themes 24
4.2.2 Presentation of Themes 25
4.2.3 Theme 1: Requirement Completeness Deficiencies 25
4.2.4 Theme 2: Requirement Ambiguity and Interpretation Variability 27
4.2.5 Theme 3: Timing and Threshold Specification Gaps 28
4.2.6 Theme 4: Requirements Quality Governance & Change Management 29

4.2.7 Theme 5: Practitioner Assessment of Requirements-Flakiness Relationship Significance . . . 31

5 Discussion **33**
5.1 RQ1: How do industry practitioners perceive the relationship between requirements quality and the flakiness of test cases? . . . 33
5.2 RQ2: Which requirements quality attributes most significantly contribute to test flakiness? . . . 35
5.3 RQ3: What guidelines can be derived from practitioner experiences to enhance requirements quality and mitigate test flakiness? . . . 40
5.3.1 Requirements Completeness . . . 40
5.3.2 Requirements Unambiguity . . . 41
5.3.3 Other Practical Guidelines . . . 42
5.4 Integration of Mixed-Methods Findings . . . 43
5.5 Theoretical Contributions . . . 43
5.6 Threats to Validity . . . 44
5.7 Future Work . . . 46

6 Conclusion **47**

Bibliography **51**

A Appendix **I**
A.1 Survey Questions . . . I
A.2 Interview Guide . . . IV

List of Figures

1.1 Decision flow illustrating flaky test scenario[26] 1

2.1 An example of a flaky test caused by asynchronous timing 9

4.1 Practitioners' experience with flaky tests and the influence of requirements 21
4.2 Impact of Requirement Quality on Flaky Tests 22
4.3 Likelihood of Requirement Issues Causing Flaky Tests 22
4.4 Estimating proportion attributed to Requirements 23
4.5 Visual representation of the codes organized into core themes 26

List of Tables

4.1 Survey Respondent Demographics 20
4.2 Overview of Interview Participants 25

1

Introduction

1.1 Problem Description

Software testing is a crucial aspect of the software development life cycle (SDLC), helping to uncover defects, enhance software quality, and ensure reliability and robustness. For example, Google executes 50 million test cases daily to ensure the quality of its products [23]. To achieve quality, many processes have become automated, and testing has followed this trend, with test cases increasingly automated to deliver better quality outcomes. However, a significant challenge within this process is flaky tests, and it becomes a major problem in test automation [35]. Flaky tests are the tests that, for the same version of code and tests, can pass and fail unpredictably on different runs [25]. Figure 1.1 illustrates a typical flaky test scenario, shows how a test is executed and categorized. A test that passes on the first run is marked PASS. If it fails initially, it is rerun up to five times; a later pass indicates a FLAKY test, while repeated failures are classified as UNEXPECTED. This behavior creates uncertainty about the reliability of the software, as it is difficult to determine whether this behavior is caused by code defects or influenced by other factors [37].

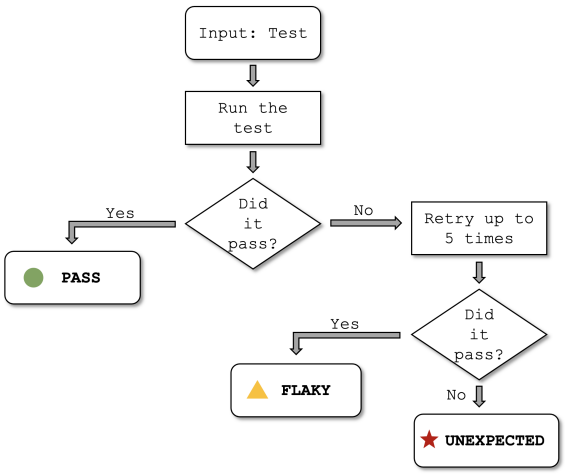

Figure 1.1: Decision flow illustrating flaky test scenario[26]

Research indicates that many software failures can be traced back to issues in re-

quirements engineering (RE), with problems arising from inconsistent, incomplete, or hidden requirements [8, 20, 27, 12, 45]. For instance, the Heathrow Terminal 5 Baggage System resulted from unclear and incomplete requirements , while the Ariane 5 rocket explosion system failure stemmed from constantly changing requirements [42].

In this context, RE refers to the creation of requirements artifacts, such as systematic specifications, use cases, or user stories that guide other development activities. Therefore, the quality of requirements has a direct impact on the success of the development process [41]. For example, ambiguous requirements can lead to incorrect or incomplete implementations [22].

Building on prior literature [8, 22], it is reasonable to hypothesize that poor requirements quality attributes, such as incompleteness, ambiguity, inconsistency , and incorrectness can contribute to flaky test behavior [4, 32, 27]. When requirements are ambiguous or incomplete, test designers make assumptions about expected system behavior [16]. For instance, vague performance requirements (e.g., the system should respond quickly) can result in timing-dependent tests that fail unpredictably due to environmental variations. Similarly, incomplete functional requirements may lead to tests that check for behaviors not consistently implemented, causing intermittent failures when the system's actual behavior varies from the tester's assumptions.

Most of the literature studies were focused on technical causes of flakiness, including concurrency, test order dependency, network issues, randomness, platform dependency, environment dependency, external behaviors, timing, and others [37]. Current solutions focus primarily on fixing technical aspects, improving test tools, or adjusting test environments [19, 54, 50]. However, despite the recognized connection between poor requirements and unstable test behavior, there has been limited systematic investigation into how software requirements quality specifically contributes to test flakiness. For example, research [4] highlights that the lack of clear requirements during the test design process is mentioned as the perceived cause of the increase in test flakiness in companies. While this research points out requirements quality as a contributing factor, it does not provide a thorough systematic analysis of this relationship.

1.2 Purpose of the Study

The lack of understanding around how requirements quality influences flaky tests limits the software engineering community's ability to implement proactive strategies that address flakiness early in the SDLC. Reactively addressing flaky tests during testing or debugging is often more costly and time-consuming than preventing them through improved requirements engineering practices. This study aims to fill that gap by investigating how industry practitioners perceive the relationship between requirements quality and the occurrence of flaky tests. It also seeks to identify which specific aspects of requirements quality most significantly contribute to flakiness in real-world software projects. Furthermore, the study aims to derive

evidence-based guidelines from practitioner experiences that can help to improve requirements quality and mitigate test flakiness in practice. Through quantitative and qualitative analysis of practitioner responses, the research seeks to uncover real-world challenges, experiences, and resolution strategies related to managing flaky tests, with the broader goal of informing better practices for requirements engineering and test design in industrial contexts. By bridging the gap between requirements specification and test reliability, the study contributes to a deeper understanding of how improving requirements quality can reduce test flakiness and enhance overall software quality assurance.

The study aims to answer the following research questions:

RQ1: How do industry practitioners perceive the relationship between requirements quality and the flakiness of test cases?

RQ2: What specific attributes of requirements quality most significantly contribute to test flakiness?

RQ3: What guidelines can be derived from practitioner experiences to enhance requirements quality and mitigate test flakiness?

This study was conducted in collaboration with the industrial partner TestScouts AB. To address these questions, a mixed-methods approach was employed, with quantitative and qualitative analysis of surveys and interviews with software professionals across different domains, aiming to identify patterns and insights about requirements-related causes of flaky tests. The findings are intended to guide improvements in requirements engineering and testing practices, enhancing overall software quality and reducing test instability.

1.3 Significance of the Study

This study is significant because it addresses a critical yet often overlooked factor affecting software quality: the connection between requirements quality and test flakiness.While prior work has focused on the technical causes of flaky tests [33, 37, 18, 36], this research explores the issue from a requirements perspective, offering a proactive approach to improving test reliability.

This connection is fundamental because requirements form the foundation for test design and expected behavior specifications [43]. When requirements are ambiguous, incomplete, or inconsistent, the test designers are forced to make assumptions about system behavior, leading to tests that may pass or fail unpredictably based on these assumptions rather than actual system correctness [22]. By improving requirements quality at its source, organizations can potentially prevent flaky tests from occurring rather than merely fixing them after they occur.

Importantly, this study not only identifies practitioner perceptions (RQ1) and key requirements quality attributes contributing to flakiness (RQ2), but also derives

evidence-based guidelines from practitioner experiences (RQ3). These guidelines provide actionable recommendations for enhancing requirements quality and mitigating test flakiness in practice, bridging the gap between empirical understanding and practical implementation. By capturing real-world experiences and transforming them into structured guidance, the study contributes both to academic research and industrial practices, supporting the development of more reliable automated testing and improved overall software quality.

2

Background and Related Work

This chapter establishes the foundational concepts essential for understanding this research, providing definitions and core principles of software testing, requirements engineering, flaky tests, and their interrelationship. It also reviews the existing research literature in three key areas: research on flaky tests, studies on requirements quality, and work connecting requirements to testing outcomes. Together, these discussions create the conceptual framework for investigating the connection between requirements quality and test flakiness. The review further reveals a significant gap in the systematic understanding of this relationship, which this study aims to address.

2.1 Requirements Engineering and Quality

Requirements Engineering (RE) is a systematic, iterative, and disciplined approach to eliciting, analyzing, specifying, and validating the requirements of a software system, ensuring that all stakeholders share a common understanding [43]. There are two main types of requirements: **user requirements**, which are are high-level and abstract descriptions of what the system should do from the user's perspective, and **system requirements**, which are detailed and technical specifications that describe the services the system must provide and the constraints under which it must operate, reflecting the needs and expectations of users [43].

A **requirements specification document** serves a wide range of stakeholders, including senior management who fund the project as well as engineers responsible for designing, implementing, and testing the system [31]. The level of detail included in these documents often depends on the type of system being developed and the software development methodology adopted.

The quality of documentation produced during the requirements phase is critical to the overall success of a software system. Poorly defined requirements can lead to misunderstandings, implementation errors, and project delays. Imprecise requirements specifications may also result in disputes between customers and developers regarding the system's expected functionality and behavior [31].

Furthermore, high-quality requirements help minimize the risk of defect propagation

into later stages of the software development life cycle, such as design, construction, testing, and deployment [22]. Effective requirements should be **documented, actionable, measurable, testable, traceable to business objectives, and specified in sufficient detail** to support system design [43].

2.2 Requirements Quality Attributes

Requirements comprises a variety of artifacts that capture what a system should do. The commonly used requirement artifacts such as user stories, system specifications, and use cases, which are integral inputs for subsequent phases of the Software Development Life Cycle (SDLC) [22, 34]. When referring to requirement quality, this study focuses on the quality of these documented artifacts. Several criteria are widely used to evaluate requirement quality. According to Lauesen [31], a high-quality requirements specification should fulfill essential criteria such as being **correct, complete, unambiguous, consistent, modifiable, verifiable, and traceable**. Similarly, the ISO/IEC/IEEE 29148:2018 outlines that each system requirement should exhibit specific attributes: it must be **appropriate, unambiguous, complete, singular, feasible, verifiable, correct, consistent, comprehensible, and capable of validation** [3]. Among the various quality attributes identified in the literature, the four attributes: **completeness, unambiguity, consistency, and correctness**are particularly prominent due to their significant influence on downstream development processes [34, 7, 6]. These attributes form the focus of this research because of their critical impact on software development outcomes.

Completeness refers to the extent to which a requirement includes all necessary information to fully describe its intended functionality or condition [3]. A complete requirement contains all essential details needed for implementation and testing, including: [31]

- Functional behavior descriptions
- Input and output specifications
- Business rules and constraints
- Dependencies on other requirements
- Edge cases and exception handling
- Performance and quality criteria.

Missing elements such as business rules, dependencies, or edge cases can result in incomplete implementations or inadequate testing. In Agile methodologies, completeness is often interpreted as a user story being "ready," including acceptance criteria and relevant details [31, 3].

Unambiguity means that the requirements allows only one interpretation [3]. An unambiguous requirement uses precise language that cannot be misunderstood by different stakeholders. Ambiguous or vague wording such as fast, efficient, or user-friendly can lead to inconsistent implementation and test results. In requirements engineering, ambiguity is described as a requirement that does not have a clear single

meaning [17, 45, 32]. Eliminating ambiguity ensures that all stakeholders, including developers and testers, share a common understanding of the requirement [8, 3]. This is particularly important in automated testing, where precise, deterministic criteria are essential for consistent test execution.

Consistency means that requirements do not conflict with one another or with other project documents [3]. In existing literature, inconsistency in requirement engineering is described, in which specifications contain conflicting or contradictory statements of the expected behavior of the system [44]. Consistent requirements maintain logical coherence across the entire specification, avoiding contradictions that could lead to [30, 1]:

- Incompatible system behaviors
- Duplicated development effort
- Logical contradictions in test design
- Dependencies on other requirements
- Conflicting implementation decisions

Correctness means that a requirement accurately captures the stakeholders' needs and intended system functionality [3]. Correct requirements align with business goals, are feasible within constraints, and do not misrepresent user expectations. Incorrect requirements often result in rework and poor user satisfaction [22, 31].

Research has revealed that requirements quality significantly impacts software development outcomes [51]. Studies shows that incomplete and incorrect requirements are among the most prevalent quality issues [12, 6, 27] representing 23.5% and 35.3% of observed defects respectively, while inconsistent requirements account for 5.9% [9]. Researchers also observed that improving the consistency of requirements can reduce completeness [29]. Furthermore, systematic study consistently reports ambiguity as one of the most common defects during requirements validation and consider it an inherent limitation of natural language specifications [9, 7]. This body of work establishes the practical significance of requirements quality issues in real-world software development.

2.3 Software Testing Fundamentals

Software testing is a critical component of the software development life cycle, serving as the primary mechanism for quality assurance and defect detection. It is essential to verify that a system behaves as intended and meets specified requirements [2, 43]. According to the IEEE standard, software testing is defined as the process of analyzing software to detect differences between existing and required conditions (i.e., defects/errors/bugs) and to evaluate the features of the software item [2].

The evolution of software development methodologies has significantly influenced testing practices. In traditional waterfall approaches, testing was typically conducted in distinct phases following the completion of development activities. In contrast, modern Agile and DevOps methodologies have integrated testing throughout the

development cycle [43]. This paradigm shift has led to an increased emphasis on automation testing to support rapid iteration and continuous delivery (CI/CD) [13]. Automated test suites are expected to execute reliably and provide consistent feedback throughout the development process. In a CI/CD environment, code changes automatically trigger the execution of these test suites, which must reliably assess software quality without human intervention. However, despite their advantages, automation frameworks face several challenges, one of the most significant being the prevalence of flaky tests, which undermine test stability and reliability [25].

2.4 Flaky Tests

Flaky tests are test cases that exhibit non-deterministic behavior, passing or failing intermittently without any modifications to the source code or execution environment [33]. This unpredictability undermines the reliability of test results and complicates software quality assurance. Recent literature reflects a growing body of research on flaky tests, with numerous new approaches emerging [36, 14, 24, 50].

Flaky tests pose a fundamental challenge to software testing by introducing uncertainty into the testing process. When a test fails, developers must determine whether it signals a genuine defect or merely reflects non-deterministic behavior [40]. This ambiguity reduces trust in automated testing, potentially leading to overlooked defects or wasted effort investigating false failures [35]. Test automation is therefore significantly impacted by the emergence of flaky tests [25].

A concrete example of a flaky test caused by asynchronous timing is discussed by Rahman et al. [36] in Fig. 2.1. The testCustomBufferSize, an email dispatch occurs in a separate thread after an error is logged. If this operation exceeds the five-second wait in the main thread, the test intermittently fails, demonstrating how non-deterministic thread execution can produce timing-related flakiness despite correct logic.

The effects of flaky tests extend beyond individual cases, disrupting entire development workflows. They can trigger intermittent failures in continuous integration pipelines, delay software releases, reduce developer productivity, and weaken confidence in automated quality assurance processes [40].

Prevalence and Impact: Research has established that flaky tests are a widespread industry problem with significant impacts on software development productivity [11, 37, 19, 18]. Studies show that flaky tests reduces trust in test results, leading developers to potentially ignore genuine failures [25]. They impose substantial maintenance burdens and frequently delay software releases by causing intermittent failures in continuous integration pipelines. Developers productivity is also reduced, as time is diverted from feature development to test debugging. At scale, these inefficiencies can result in measurable economic costs; for example, Microsoft has reported losing thousands of developer hours annually due to flaky tests [49].

```
1  @Test public void testCustomBufferSize() {
2    startSMTPServer(NO_SSL);
3    configure(...);
4    logger.error("hello");
5    waitUntilEmailIsSent();
6    MimeMultipart mp = verifyMultipart("..." + this.
       getClass().getName() + " - " + msg);
7  }
8  MimeMultipart verifyMultipart(...) {
9    waitToReceiveEmails(1);
10   assertEquals(1, server.getMessages().length);
11 }
12 class SenderRunnable implements Runnable {
13   void run() { sendBuffer(...); } // sends e-mail
14 }
```

Fig. 1. Example TD test from `SMTPAppender_GreenTest` class in the `logback` project [38].

Figure 2.1: An example of a flaky test caused by asynchronous timing

Technical Causes Identified: Initial research on flaky tests focused predominantly on technical factors. Luo et al. conducted an empirical analysis identifying ten distinct root causes, primarily including implementation-level issues such as asynchronous waits, concurrency problems, resource leaks, test order dependency, network issues, timing problems, I/O dependencies, randomness, floating point operations, and unordered collections [33]. Thorve et al. extended this foundation by analyzing Android projects, adding causes related to program logic, dependencies, and UI behaviors [48]. Eck et al. surveyed developers to explore their perceptions of flaky test causes, confirming that most issues were rooted in technical implementation [18].

Contemporary Research: Recent studies have continued to focus primarily on technical causes and solutions. Amjad et al. produced a comprehensive taxonomy emphasizing concurrency, test order dependency, network availability, and randomness [37]. Habchi et al. conducted qualitative studies confirming these technical focuses while noting that the analysis of flaky tests must consider the whole testing ecosystem [25].

2.5 Requirement Engineering and Software Testing

Software testing and requirements engineering are interrelated disciplines that complement each other in ensuring software quality. According to IEEE standards, requirements form the foundation for test design, while testing serves to validate that the implemented system meets those requirements [53]. Requirements are the foundation on which the software systems are built. There are many failures in software systems stem from poor requirements definition [8], the relationship between requirements and testing is bidirectional [41]:

Requirements → Testing: Requirements serve as the basis for:

- Test case design and test scenario creation
- Test data specification and boundary condition identification
- Acceptance criteria definition
- Test coverage determination

Testing → Requirements: Testing activities provide:

- Validation of requirements feasibility and clarity
- Feedback on requirements completeness and consistency
- Discovery of missing or incorrect requirements
- Evidence of requirements satisfaction

Effective coordination between Requirements Engineering and Software Testing is essential for delivering high-quality software. A misalignment between these disciplines significantly reduces the likelihood of project success, as both disciplines play a crucial role in ensuring that the developed product meets customer expectations in terms of functionality and quality [47]. Research on the international standardization of software quality and testing emphasizes the importance of aligning requirements and testing practices to achieve consistent and reliable outcomes [53]. **Testability** emerges as a critical quality attribute that bridges requirements and testing. A testable requirement can be verified through specific test cases with objective pass/fail criteria [52]. Requirements that lack testability create challenges for validation and quality assurance. This foundational understanding of requirements quality attributes, and their relationship to testing provides the conceptual framework for investigating how specific requirements quality deficiencies contributed to flaky tests.

2.5.1 Limited Research on Upstream Flaky Tests Causes

While extensive research has examined technical causes of flaky tests, limited attention has been paid to potential upstream causes in the software development lifecycle:

Specification Nondeterminism: Shi et al. introduced the idea that flakiness can stem from specification nondeterminism, where tests implicitly assume deterministic behavior from inherently non-deterministic APIs. This work suggested that ambiguities in specifications may be an overlooked cause [39].

Requirements-Related Factors: Ahmad et al. identified upstream issues such as unclear requirements as potential contributors to flakiness, especially in organizations using ad-hoc test design approaches. However, their study did not systematically investigate specific requirements quality attributes [5].

2.6 Summary

The reviewed literature establishes that requirements quality has a significant influence on software development outcomes, and that flaky tests are a pervasive industry challenge affecting testing. However, existing research in requirements engineering largely focuses on the impact of specification quality on implementation correctness and project delivery, while flaky test research concentrates on technical and environmental causes. The intersection between these domains remains underexplored.Although prior studies have examined practitioner perspectives on flaky tests [35, 4, 37, 36, 33] and documented common challenges in achieving high-quality requirements [20, 6, 22], none have systematically investigated how specific requirement quality attributes contribute to flaky test phenomena. Addressing this gap could enable preventive strategies for flaky test reduction through improved requirements practices, the integration of explicit considerations into requirements engineering, enhanced coordination between requirements and testing activities, and more comprehensive approaches to testing reliability that address upstream causes.

.

3

Methods

3.1 Research Design : An Exploratory Case Study

Software engineering research is often categorized into two primary types: **knowledge-seeking and solution-seeking.** *Solution-seeking* research typically involves designing, creating, or developing tools, models, or algorithms to address specific challenges in software engineering. In contrast, *knowledge-seeking* research focuses on exploring and uncovering new insights, theories, or understandings. This approach is particularly valuable when investigating complex or nuanced phenomena or when establishing a foundational knowledge base for future studies [46].

In accordance with Robson's classification, Runeson et al. identified four different types of research purposes, including **exploratory research**, which aims to determine what is happening, gain new insights, and develop ideas and hypotheses for future research [38].

The approach adopted in this thesis follows a **knowledge-seeking approach**. The central aim is to investigate the relationship between flaky tests and the quality of software requirements. The study focuses on gathering insights from industry practitioners to deepen the understanding of this underexplored aspect of software engineering. This research employs an **exploratory case study** to investigate practitioners' experiences with flaky tests and requirement quality (RQ1 and RQ2). Furthermore, it seeks to integrate these practitioner experiences into actionable guidelines to enhance requirements quality and minimize test flakiness (RQ3). Mixed methods research is particularly well-suited where both the breadth of patterns (quantitative data) and the depth of individual experiences (qualitative data) are used [28]. This methodological choice enables a comprehensive exploration of how requirements quality may influence test flakiness from multiple perspectives.

The study employed a sequential mixed methods approach. First, a quantitative survey was conducted to identify general trends and patterns among a broader sample of software engineering practitioners. This provided an overview of practitioners' perceptions regarding the relationship between requirements quality and test flakiness. The quantitative data were analyzed using descriptive statistics. Following the survey, qualitative semi-structured interviews were conducted to explore the identified trends in greater depth. These interviews offered deeper insights into

practitioners' experiences and perspectives that could not be fully captured through survey responses alone. To analyze the qualitative data, thematic analysis was used to identify recurring patterns, concepts, and relationships within participants' responses. This sequential approach provided richer insights than would have been possible through a single method.

3.2 Data Collection

To understand how industry practitioners perceive the relationship between requirements quality and the flaky tests,this study uses a combination of quantitative and qualitative data collection methods. These methods were chosen to identify the initial pattern and generate detailed insights. The study design directly supports the research questions: the survey primarily addresses **RQ1** and **RQ2**, while the interviews provide richer explanations for these questions and contribute to the development of actionable guidelines in **RQ3**.

3.2.1 Quantitative Study: Survey

A survey is a type of empirical study in which data is collected by asking questions to participants. It provides quantitative or numerical descriptions of trends, attitudes, or opinions within a population by studying a sample of that population [15]. The survey aims to investigate the relationship between flaky tests and the quality of the requirements by answering the research questions. For RQ1, it captures practitioners' overall perceptions of how requirements quality influences test flakiness. For RQ2, it systematically examines specific requirements quality attributes to determine which ones practitioners consider most significant in contributing to flaky tests.

Survey Instrument

The survey instrument was constructed as a structured questionnaire, designed to ensure clarity, relevance, and ease of understanding. In designing the survey, special attention was given to clarity through carefully phrased questions to minimize ambiguity and to relevance, by ensuring that only questions directly linked to the research objectives were included. It included 10 closed-ended questions. Closed-ended questions primarily used Likert scales to quantitatively capture participants' perceptions of flaky tests and requirements quality attributes, including unambiguity, completeness, correctness, and consistency. These attributes were chosen because prior studies have frequently identified them as critical factors influencing the clarity and reliability of software specifications [8, 45, 44, 27]. Furthermore, ambiguity, incompleteness, and incorrectness in requirements have been consistently associated with difficulties in both development and testing activities, making them particularly relevant for exploring their potential impact on test flakiness [17, 6].

The questionnaire underwent a validation process: it was reviewed by university and Test Scouts supervisors, followed by five academic experts who assessed clarity, relevance, and appropriateness. Revisions were made based on their feedback. Before

full deployment, a pilot study was conducted to evaluate the survey's clarity and effectiveness. Feedback from two initial participants indicated that two questions required clarification. These were revised and tested with three additional participants, whose consistent responses confirmed that the issues were resolved. This process ensured the survey instrument was valid, reliable, and ready for broader distribution.

Survey Population

The survey target audience consisted of professionals involved in various phases of software development and testing, including software testers, developers, and requirements analysts. Participation was also open to individuals with experience or knowledge of flaky tests and requirements, regardless of formal job titles. This broad approach aimed to capture a diverse range of perspectives from both industry and academia.

Survey Deployment

To facilitate wide participation, the survey was distributed electronically through multiple channels. It was shared on LinkedIn, via private invitations, in online testing communities such as Discord, and on relevant blogs and forums, including Reddit. Additionally, the survey was distributed internally to the Test Scouts office group through their Microsoft Teams communication channel. This multi-channel approach enabled efficient, global data collection over the survey period.

3.2.2 Qualitative Study: Semi-Structured Interviews

The most widely used method in qualitative research is **interviews**. There are three types of interview processes: structured interviews, unstructured interviews, and semi-structured interviews. *Structured interviews* involve asking a list of predetermined questions, with minimal variation between interviews. No follow-up questions are permitted based on the responses, ensuring consistency across participants. This approach enables systematic data collection and facilitates comparative analysis. In contrast, *unstructured interviews* gravitate toward open-ended conversation and do not follow any pre-planned set of questions. This approach allows participants to express themselves freely and enables researchers to explore unanticipated themes that emerge during the dialogue. *Semi-structured interviews* combine characteristics of both approaches. They employ a predefined set of questions asked of all participants while also allowing spontaneous inquiries that emerge during the conversation. This hybrid method offers a balance between consistency and flexibility, enabling researchers to address specific research questions while remaining responsive to participants' unique perspectives and experiences [28]. For this study, semi-structured interviews were conducted to provide detailed qualitative insights for RQ1 and RQ2 while also serving as the primary basis for synthesizing practitioner-informed guidelines in RQ3. The survey identified industry patterns regarding practitioners' perceptions of the relationship between requirements quality and test flakiness (RQ1) and the different quality attributes that contribute to test flakiness (RQ2). The in-

terviews were designed to explore the underlying mechanisms and gather in-depth explanations of how requirements quality influences test flakiness based on practitioners' real-world experiences and to generate practical strategies for addressing these issues, thereby informing the development of guidelines (RQ3).

Interview Guide:

Semi-structured interviews were chosen to explore practitioners' perspectives in depth while allowing flexibility for open-ended responses. The interview guide was developed consisting of seven open-ended questions. Each interview followed this guide, incorporating the core and follow-up questions when necessary. The interview questions were formulated after analyzing trends in the survey responses to generate questions distinct from those in the survey to avoid redundancy. The primary objective of the interviews was to obtain detailed and nuanced responses on aspects not sufficiently captured through the survey. The interview questions were strategically developed based on key trends identified in the survey responses. Some questions directly explored the connection between requirements and flaky tests, as survey data suggested this relationship but lacked specific examples and mechanisms. Dedicated questions investigated requirements quality attributes, prompted by survey responses indicating concerns about requirements clarity while othes aim to discover practical experiences, coping strategies, and improvement practices. These responses formed the foundation for deriving the actionable guidelines presented in the discussion (RQ3), enhancing the validity of findings through methodological triangulation. Finally, open-ended questions were incorporated to create space for unanticipated insights, addressing a limitation of the structured survey format. This complementary approach allows the research to achieve both breadth (survey) and depth (interviews). The guide was reviewed and validated by university supervisors, the industry partner, and four other academic experts for clarity, comprehensibility, and logical structure. Revisions were made based on feedback, and three pilot interviews were conducted to refine the guide before final deployment.

Interview Population

The strategy for recruiting participants in this study was based on convenience sampling [28]. The goal was to recruited individuals who were willing to participate and possessed relevant expertise to provide meaningful insights into the topic. A total of eight participants were selected. All participants, both internal (from Test Scouts) and external, were selected voluntarily. Four internal participants were available, while no additional Test Scouts members could participate due to prior commitments. To complement this, four external participants were recruited from survey respondents who had experience with flaky tests and were willing to take part in follow-up interviews. Including external practitioners ensured a diversity of perspectives and helped mitigate potential organizational bias, capturing a broader range of experiences and practices beyond a single company context.

Interview Deployment

To systematically capture and analyze insights, the interviews were audio recorded. Each interview lasted between 30 and 50 minutes. Prior to recording, participants were informed about the purpose of the study and provided their consent. They were also given explicit information regarding the procedures for ensuring anonymity and confidentiality. The interviews with members of the Test Scouts team were conducted in their office in Gothenburg, while the interviews with external practitioners were carried out via virtual team meetings. All interviews were automatically transcribed using Teams' transcription tool. To ensure the answers were accurately recorded, each recording was manually reviewed, and errors introduced by automated transcription were corrected.

3.3 Data Analysis

To comprehensively address the research questions, this study employed multiple approaches to analyze the data, reflecting its mixed-methods design. Both quantitative and qualitative data were systematically analyzed to capture overall trends as well as detailed aspects of the relationship between flaky tests and requirements quality.

3.3.1 Quantitative Analysis

Quantitative data were collected through a survey, resulting in 85 responses via Microsoft Forms. After removing incomplete, duplicate, and inconsistent entries, 74 valid responses were retained for analysis. The Likert scale data then underwent a preparation process in which responses were converted into numerical values. Missing values were observed, primarily due to incomplete responses from some participants, and these were excluded from the analysis to avoid distorting the final results.

Descriptive analyses were conducted to explore and summarize the quantitative survey data. Using Microsoft Excel, the data were organized and visualized through tables and charts. Measures such as frequencies, percentages, means, and standard deviations were calculated to provide an overview of participants' responses and to identify general trends regarding their perceptions of flaky tests and requirements quality. Descriptive statistics were selected because they offer a clear yet robust method for characterizing the dataset, enabling the identification of key patterns, distributions, and variations within the responses.

3.3.2 Thematic Analysis

The qualitative data was gathered through eight semi-structured interviews with industry practitioners. To analyze the qualitative data, this study employed a Braun and Clarke thematic analysis six-step approach to systematically identify, interpret, and report themes within the dataset [10]. This method facilitated an in-depth exploration of the underlying meanings reflected in participants' experiences and perspectives. To maintain rigor and ensure trustworthiness, the analysis followed established guidelines and was carried out through multiple iterative and reflective steps, allowing for comprehensive engagement with the data throughout the process.

Data Familiarization

First, the data were pre-processed by extracting key notes, removing duplicate entries, and standardizing expressions to ensure consistency for subsequent analysis. Following this, familiarization with the data was achieved through repeated, thorough readings of the interview transcripts, during which patterns such as recurring concepts, unclear or vague statements, and implicit assumptions were identified. This in-depth engagement provided a comprehensive understanding of the content and supported the emergence of preliminary analytical insights.

Systematic Coding

In the second phase, the data were systematically coded using an inductive coding approach. This method allowed codes to emerge directly from the data, rather than being predetermined by an existing framework. Meaningful text segments that encapsulated core ideas were labeled accordingly. Inductive coding reduced the complexity of the raw data and highlighted patterns relevant to the research questions. These codes subsequently served as the foundation for the development of broader themes, facilitating the extraction of nuanced insights.

Theme Development

This step involved integrating the codes into themes and identifying patterns and relationships within the data to address the research questions. Themes were conceived as higher-level summaries of the codes, capturing not only the extracted elements of the data but also meaningful explanatory concepts. After completing keyword extraction and coding, codes with similar meanings were grouped together, and representative themes and sub-themes were developed through repeated comparison and verification.

Theme Refinement and Review

The subsequent phase involved reviewing and refining the themes to ensure internal consistency and clarity. Each theme was evaluated against both the coded extracts and the complete dataset to confirm that it accurately represented participants' narratives. Particular emphasis was placed on verifying that the themes were well-supported by the data and aligned with the central research questions.

Defining and Naming Themes

In the final stage, each theme was precisely defined and assigned a clear, meaningful label that encapsulated its core essence. These thematic definitions formed coherent narratives, which served as the foundation for the analytical discussion presented in the findings section.

4

Results

This chapter presents findings from a mixed-methods study consisting of 74 survey responses and eight semi-structured interviews with industrial practitioners. A survey was conducted to identify patterns and perceptions regarding the relationship between flaky tests and requirements, whereas semi-structured interviews were subsequently conducted to explore identified patterns in greater depth. These interviews enabled a deeper understanding of industrial perspectives on flaky test causes and their relationship to requirements quality attributes.

4.1 Survey Findings

This section presents the quantitative findings derived from a survey conducted with 74 industry practitioners. The data collected was systematically analyzed using descriptive statistics to summarize responses and provide an overview of key variables relevant to the relationship between requirements quality and test flakiness.

A comprehensive understanding of the survey participants' backgrounds is crucial for contextualizing the findings and assessing their generalizability. The survey successfully gathered responses from individuals across diverse roles, experience levels, and industry sectors, thereby establishing a robust dataset for analysis.

The participant distribution shows a balanced mix of experience levels that can be shown in Table 4.1, with 34% having over 10 years and 30% having 0–3 years in the field. This diversity suggested that practitioners with extensive experience can provide insights based on long-term industry patterns. The survey participants represented various industry sectors, with the majority (55%) originating from the software development industry. The automotive sector accounted for 26% of respondents, with smaller proportions from finance, e-commerce, and healthcare. Regarding involvement in the Software Development Life Cycle (SDLC) phases, the largest proportion of participants (96%) reported being primarily involved in the testing phase. This was followed by involvement in requirements (26%) and development (22%). It is important to note that this was a multiple-selection question, allowing participants to indicate engagement in more than one SDLC phase.

Having established the diverse backgrounds and substantial experience of the survey

Table 4.1: Survey Respondent Demographics

Aspect	N=74	Percentage
Work Experience		
0–3 years	22	30%
4–6 years	17	23%
7–10 years	10	14%
10+ years	25	34%
Involvement in SDLC Phases		
Requirements	19	26%
Design	4	5%
Development	16	22%
Testing	71	96%
Deployment & Maintenance	11	15%
Industry Sector		
Tech & Software Development	41	55%
Automotive	19	26%
Finance & Banking	8	11%
E-commerce and Healthcare	2	2%
Other	4	5%

participants, the analysis now turns to the core research questions guiding this study. The investigation is structured around two main research questions. For clarity, 'SQ' refers to individual Survey Questions from the questionnaire. RQ1 explores how industry practitioners perceive the relationship between requirements quality and test flakiness, drawing on responses to SQ5, SQ6, and SQ10. RQ2 investigates which specific attributes of requirements quality are most likely to contribute to flaky tests, informed by SQ7, SQ8, and SQ9. Additionally, SQ4 provides contextual insight into the general prevalence and frequency of flaky test experiences across the participant pool. The following sections present these findings in relation to each research question.

The survey included five Likert-scale questions: SQ4, SQ5, SQ7, SQ8, and SQ9 using a scale ranging from Always to Never. In addition, SQ6 and SQ10 were presented as single-choice questions to capture specific practices and estimations related to requirements-driven test instability.

The survey results demonstrated that flaky tests are a prevalent issue within the industry. When participants were asked *how often they experienced flaky tests (SQ4)*, a substantial 80% of the 74 surveyed practitioners reported encountering flaky tests in their projects. This finding is further underscored by the fact that only one respondent (1%) reported never experiencing flaky tests, establishing this phenomenon as a nearly universal challenge in contemporary software development practice. A majority of practitioners (56%) reported *frequently referring to requirements(SQ5)* 4.1, indicating a general recognition that requirements are a relevant factor in under-

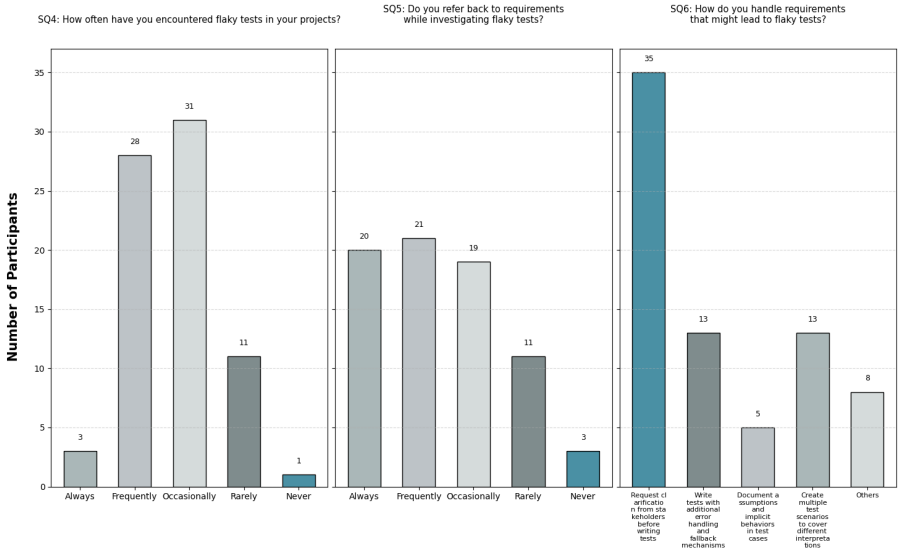

Figure 4.1: Practitioners' experience with flaky tests and the influence of requirements

standing or resolving test flakiness. However, a significant proportion (14%) reported rarely consulting requirements during such investigations, and three participants(4%) indicated never doing so. This divergence between acknowledging the relevance of requirements and consistently integrating them into daily investigative practices is a critical observation suggesting that while awareness of the relationship exists, its practical application may be hindered.

Practitioners employ a variety of strategies to mitigate test flakiness related to requirement specifications. When asked *how they handle requirements they believe lead to flaky tests (SQ6)*, the most common approach reported by 48% of respondents 4.1 was seeking clarification from stakeholders. Other strategies include adding error handling and designing multiple test scenarios with 18% of respondents each. These findings suggest that practitioners are aware of the impact that ambiguous or unstable requirements can have on test flakiness and take proactive measures to address such issues, although the predominance of clarification seeking indicates that many requirements related problems stem from ambiguity rather than technical complexity.

The industry's recognition of the relationship between requirements quality and test flakiness is evident in the survey responses addressing specific requirement quality attributes 4.2. A significant majority of respondents (80%) indicated that requirements completeness frequently cause flaky test, with only 1% never observing this connection. This strong consensus among practitioners signifies a clear perception of a robust connection between the quality of requirements, particularly in terms of their clarity and completeness, and the consistent, reliable behavior of tests. Similarly, a substantial number of participants also reported a frequent association between flaky tests and poorly defined test conditions. Specifically, 6 participants (8%) experienced this issue, while 46 (62%) encountered it with moderate to high regularity. Only 14

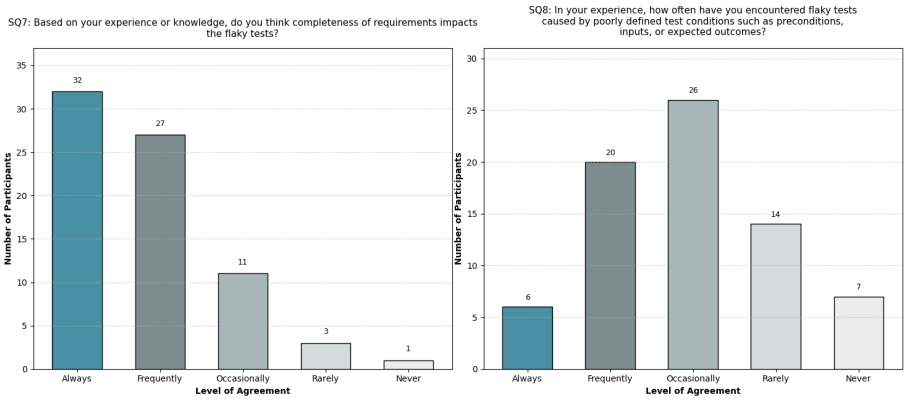

Figure 4.2: Impact of Requirement Quality on Flaky Tests

(19%) noted it as an infrequent occurrence, whereas 7 (9%) participants reported never encountering this issue. These figures collectively underscore the widespread prevalence of insufficiently specified test conditions, a problem that often originates either from flaws in the test design process itself or, more fundamentally, from a lack of clear and complete requirements.

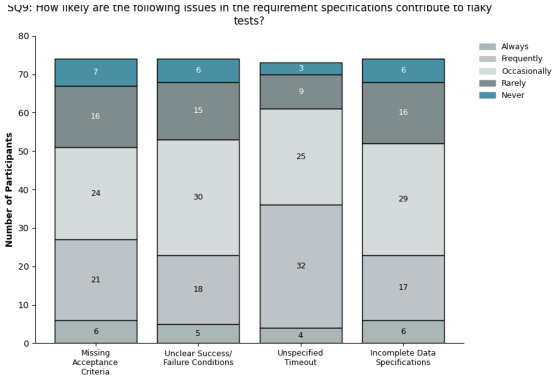

Figure 4.3: Likelihood of Requirement Issues Causing Flaky Tests

The survey findings, as illustrated in Figure 4.3 ,indicate that practitioners perceive all four categories of requirement specification issues as contributing factors to test flakiness, albeit with varying levels of reported impact. **Unspecified timeout durations** emerged as the most significant issue, reported by 77% of respondents as a cause of flakiness, with only 12% indicating they rarely encounter this problem. This high percentage suggests a significant influence of missing or implicit timing constraints within requirement specifications. **Unclear success and failure conditions** were identified by a majority (65%) of participants as relevant contributors to flakiness, whereas 8% never encountered this issue. This highlights the critical role of ambiguity and a lack of precision in defining the expected outcomes of system behavior. Similarly, a majority of participants (61%) also identified the**absence of clear acceptance criteria** as a relevant contributing factor. This reinforces the necessity for well-defined benchmarks against which system functionality can be verified. **Incomplete data specifications**, cited by 46 participants (62%)as con-

tributing to flaky test behavior. This further underscores the negative impact of incomplete or vague data requirements on the reliability and consistency of tests.

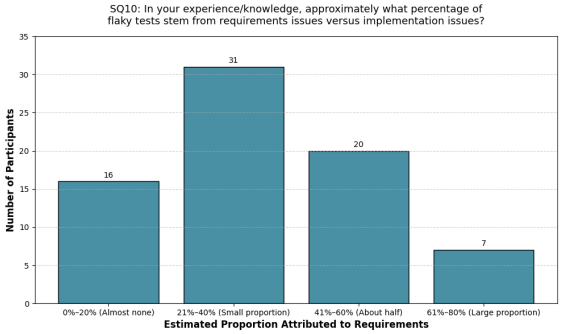

Figure 4.4: Estimating proportion attributed to Requirements

To understand practitioners' nuanced views on the extent to which flaky tests originate from requirements issues versus other factors, such as implementation-related problems, we included question SQ10. The participants expressed diverse views , While responses were distributed across all ranges, most participants estimated that a moderate proportion of flaky tests (between 21% and 60%) stems from requirements related problems. Only a smaller subset attributed either none (0%–20%) or a large proportion (over 60%) of flaky tests to requirements. This distribution suggests that although there is no universal agreement, a significant number of practitioners perceive requirements issues as a moderate but substantial contributor to test flakiness.

Survey Conclusions

Having established the diverse backgrounds and substantial experience of the survey participants, the analysis now turns to the core research questions guiding this study. Contextual insight from SQ4 demonstrates that experiences of flaky tests are common across the participant pool, underscoring the relevance and importance of examining the relationship between requirements quality and test flakiness. RQ1 explores how industry practitioners perceive the relationship between requirements quality and the flakiness of test cases. Responses to SQ5, SQ6, and SQ10 indicate that participants generally perceive a strong connection between these two factors. Most practitioners perceive that a moderate proportion of flaky tests (21%–60%) originate from requirements issues (SQ10), indicating they consider requirements a contributing factor to test instability. Building on these perceptions, RQ2 investigates which specific attributes of requirements quality most significantly contribute to test flakiness, drawing on responses from SQ7, SQ8, and SQ9. The survey results revealed primary findings that informed the subsequent semi-structured interviews. Practitioners generally recognize that requirements quality, particularly clarity and completeness, has a direct impact on test stability (SQ7). However, this awareness is not consistently applied in practice, with only 56% frequently consulting requirements during flaky test investigations (SQ5). This gap between recognition and practice represents a significant finding for understanding current industry

approaches. When addressing flaky tests arising from requirements issues, practitioners employ various approaches, with seeking stakeholder clarification being the most common response (SQ6). However, these responses are often informal or ad hoc rather than systematic. The survey identified specific contributing attributes (SQ9) that practitioners associate with test flakiness, including unspecified timeouts, unclear success conditions, missing acceptance criteria, and incomplete data specifications. These findings provided clear directions for the interview phase to explore these patterns in greater depth and understand the mechanisms by which these quality issues translate into test instability. The survey findings establish a foundation for the qualitative interview phase, with practitioners demonstrating awareness of the requirements-flakiness relationship while revealing inconsistencies in how this knowledge is applied in practice. The identification of specific quality attributes offers concrete areas for deeper investigation into the causal mechanisms underlying test flakiness.

Although the survey primarily addressed RQ1 and RQ2, its findings also highlighted issues such as unclear requirements, missing acceptance criteria, and unspecified timeouts. These insights not only informed the design of interview questions for deeper exploration of RQ1 and RQ2 but also identified key areas where practitioners need guidance, directly contributing to the development of evidence-based guidelines to address RQ3.

4.2 Thematic Analysis of Interview Data

Based on the eight interview transcripts, the final stage of analysis, conducted through multiple rounds of in-depth examination, produced 20 inductive, data-driven codes. These codes were subsequently organized into five core themes, which were mapped to the research questions and reviewed for consistency. The identified themes collectively illuminate the dynamics between flaky tests and requirements quality, revealing relationships that influence industrial practitioners' practices. While deepening the exploration of RQ1 and RQ2, the interviews simultaneously gathered practitioner insights that served as the empirical foundation for addressing RQ3. To provide context on the participants contributing to these insights, Table 4.2 presents an overview of the interviewees, including their current roles, years of professional experience, and respective industry domains. The following section provides a detailed examination of each identified theme, incorporating participant quotations as empirical evidence to support the interpretations.

4.2.1 Overview of Codes and Themes

The 20 inductive codes identified from the interviews were subsequently organized into five core themes, as illustrated in Figure 4.5. This thematic map highlights the relationships among codes and themes and their alignment with the research questions. All codes and themes were carefully reviewed and validated by the supervisor to ensure accuracy and consistency with the study objectives.

Table 4.2: Overview of Interview Participants

ID	Current Role	Years of Experience	Industry
P1	Senior Software Developer	8	Healthcare
P2	Test Architect	10	Banking
P3	Automation Engineer	7	Automotive
P4	Automation Consultant	3	Automotive
P5	DevOps Engineer	7	Software&IT
P6	Test Lead	8	Automotive
P7	System Test Developer	8	Software&IT
P8	Software Automation Tester	3	Automotive

4.2.2 Presentation of Themes

The subsequent sections detail the findings for each identified theme, incorporating illustrative quotes from the interviews to contextualize and support the analysis. These quotations provide empirical evidence, highlighting participants' perspectives and experiences.

4.2.3 Theme 1: Requirement Completeness Deficiencies

A recurring theme identified in the analysis concerned requirement completeness deficiencies that contribute to test flakiness. This theme directly connects to the survey finding that 80% of practitioners reported requirements completeness as frequently impacting test flakiness. Requirement completeness refers to the extent to which a specification contains all necessary details for accurate implementation and testing, including critical parameters and real-world operating constraints such as supported browsers and different version details. Participants reported that when requirements lacked these essential elements, test designers were forced to make assumptions, resulting in tests that behaved inconsistently across different environments or scenarios. These gaps in the requirements often led to test cases that would pass under certain conditions but fail unpredictably, increasing overall test flakiness and undermining confidence in automated testing suites.

The impact of insufficient specification became evident through participant experiences. One participant emphasized the challenge, stating:

> *Most of the time, flaky tests occur because the requirements are not specific enough. The requirements do not include detailed pre-conditions for the test, and that changes the result of the test if I only assume them. —P3*

This observation highlights how insufficient specification of preconditions can introduce variability in test execution, where different assumptions about initial conditions can lead to inconsistent test outcomes.

Participants also pointed out that even when requirements appear detailed, missing

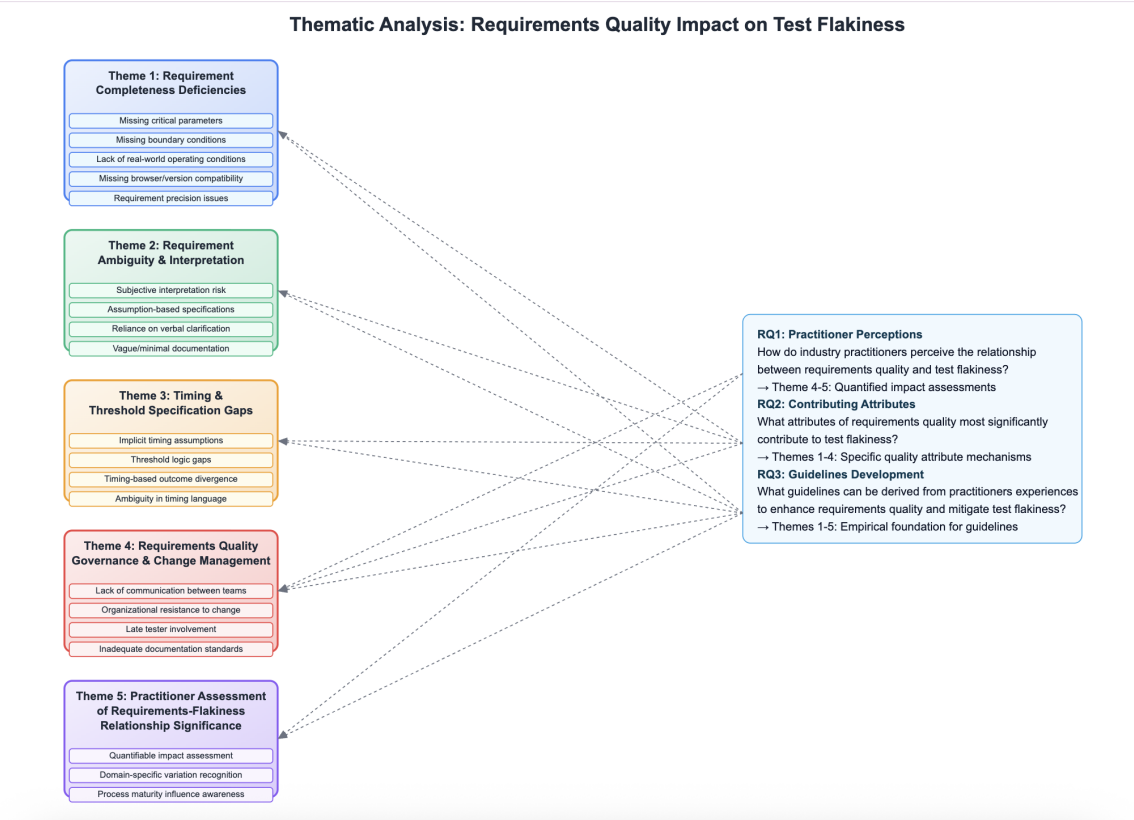

Figure 4.5: Visual representation of the codes organized into core themes

critical parameters can cause significant flaky tests issues:

> *Requirements defined a specific calculation method but omitted two critical parameters, leading to inconsistent outcomes when those parameters varied. —P6*

Beyond missing parameters, the absence of real-world operating context, such as supported browsers or platform versions, further addresses the problem.One participant reflected on this, saying,

> *Flaky tests occurred due to missing requirements about supported browsers and platform compatibility, as a results JavaScript version caused inconsistent behavior across different browsers. —P2*

This reflects a gap not only in documented requirements but also in tester assumptions about browser coverage. Testers primarily validated the system on common browsers, where tests passed consistently. When the same tests run across a wider range of browsers, previously unseen compatibility issues emerged, resulting in flakiness and undermining trust in the automation suite.

Overall, these findings demonstrates that incomplete requirements not only increase the risk of flaky tests but also slow down development and testing cycles due to rework and troubleshooting. These findings collectively demonstrate how requirements

incompleteness creates information gaps. Such omissions contribute to unpredictable test behavior and hinder the development of stable, reliable tests,directly supporting the survey finding that incomplete data specifications affect 62% of practitioners. By identifying completeness deficiencies as a significant contributor, this theme provides a concrete answer to RQ2, and this insight directly informed the practitioner guidelines in Chapter 5 to answer RQ3, which emphasize completeness as an important factor for reducing flakiness.

4.2.4 Theme 2: Requirement Ambiguity and Interpretation Variability

Ambiguity in requirements emerged as a significant concern among participants, directly aligning with the survey finding that 65% of practitioners identified unclear success and failure conditions as contributors to flakiness. Vague or overly flexible language left room for multiple interpretations, often resulting in unstable or misaligned tests. This issue was particularly critical in automated testing environments, where precise and testable conditions are essential.

Tne participant shared:

> *The test generated random names with varying lengths, sometimes four digits, sometimes five or three. The test passed only when the name length accidentally matched the expected format. The requirement mentioned a format but didn't define it, which led to different interpretations. —P2*

This demonstrates how format ambiguity can create tests that pass or fail based on random chance rather than genuine system behavior, representing a classic manifestation of flakiness rooted in requirements quality issues. Such unclear requirements forced testers to guess implementation details, increasing test unpredictability.

Terms such as "fast," "comfortably," or "as soon as possible" were frequently cited as problematic due to their lack of objective, measurable criteria. The participant highlighted the challenge of interpreting qualitative descriptions:

> *Some requirements don't include facts, but they include feelings, such as when you release the brake, the truck should start rolling comfortably. What's comfortable for the driver? —P3*

This observation illustrates the fundamental challenge of translating subjective requirements into objective, testable conditions, where the absence of measurable criteria makes it impossible to develop consistent test assertions. This type of subjective language made it difficult to define precise test criteria, adding to flakiness. Additionally, broad or overly general requirements also caused confusion. A participant recounted receiving a vague requirement:

> *I received a requirement that was a one-liner: 'Implement file upload functionality for a specific e-commerce application.' Now, file upload could be*

considered at many places, so it was vague. —P5

These examples collectively reveal how ambiguity and lack of detail in requirements increase interpretation variability among team members, making it challenging to design stable and reliable test cases. When requirements contain subjective language, undefined formats, or overly broad specifications, they create an environment where test outcomes become dependent on individual interpretations rather than clear, objective criteria.

This theme also aligns closely with survey findings, which identified specific attributes of requirements associated with test flakiness, including unclear success and failure conditions. The theme provides deeper insight into these patterns, illustrating how ambiguous or broadly defined requirements force testers to interpret implementation details individually, thereby increasing variability in test outcomes. Together, the survey and interview findings reinforce that ambiguity is a critical attribute of requirements quality contributing to flakiness, directly addressing RQ2. Also, the recognition of ambiguity and interpretation variability reinforced the need for precise, measurable requirements. These observations were translated into guidelines that recommend strategies for eliminating vague or subjective terminology in requirements.

4.2.5 Theme 3: Timing and Threshold Specification Gaps

This theme directly corresponds to the survey's most significant finding: 77% of practitioners reported unspecified timeout durations as a cause of flakiness. The interview analysis revealed that timing-related flakiness emerged as a distinct and significant issue, with participants identifying time-related requirement issues as a primary cause of flaky test behavior in their projects.

In most of the cases, requirements contain implicit timing assumptions, lack of explicit threshold values or ambigious specifications, leading to inconsistent results across test runs. Missing or unclear timing requirements, as well as absent threshold definitions, led to inconsistent outcomes across test runs.

For example, one participant explained:

> *If the time limits provided in the requirements are incorrect or if no time is specified we make assumptions. Based on the test results, we then adjust the code and update the timing constraints accordingly. —P8*

This highlights how the absence of defined timing parameters forces teams into reactive adjustments, increasing instability in tests and creating a cycle of assumption-based modifications rather than requirement-based precision. Such timing gaps could be subtle and difficult to detect. Another participant described a case where:

> *A student scheduling app for airplane work, stable for three years with no updates, suddenly crashed. After two months of investigation, the cause was traced to daylight saving time and regional time differences that was*

not define in requirements that led to flaky behavior. —P4

This case demonstrates how time-related edge cases, when not explicitly addressed in requirements, can cause tests to become flaky within seemingly stable systems. It also created a false sense of security, masking the underlying vulnerability to specific temporal conditions. This example underscores how requirements must anticipate and explicitly address various temporal contexts, including time zone changes, daylight saving transitions, leap years, and other calendar-related edge cases that may only manifest under specific circumstances.

These findings collectively demonstrate that clear, quantitative timing and threshold specifications are not only helpful additions to requirements but essential components for achieving stable, reliable tests. The theme reveals that time-related requirements must address not only obvious timing constraints but also edge cases, system reaction patterns. By exposing the hidden risks of implicit timing and threshold assumptions, these findings shaped guidelines that advocate for explicit temporal and threshold specifications in requirements.

4.2.6 Theme 4: Requirements Quality Governance & Change Management

The interview analysis revealed that organizational procedures and constraints often create barriers that prevent teams from maintaining good requirements quality, which participants identified as a significant factor contributing to test flakiness. This theme helps explain the gap identified in the survey between practitioners' awareness of requirements issues (80% recognizing completeness impact) and their inconsistent application of this knowledge in practice (only 56% frequently consulting requirements).

One participant stated:

There is a resistance in software development to alter requirements, especially if they have been active for quite a while. —P6

Participants reported that organizational resistance to updating requirements after they are documented often results in specifications that no longer fully match the system. When requirements can't be updated as the system evolves, they lose their accuracy, which means tests based on these old specifications become unstable and leaving testers to make assumptions or create loosely defined expected outcomes. Under varying runtime conditions, these assumptions sometimes align with the system's actual behavior (pass) and sometimes do not (fail), even when no code changes occur. Practitioners described this mismatch between ambiguous requirements and variable conditions as a common way in which poor requirements quality produces flaky tests.

One participant explained organizational constraints force teams to accept poor requirements quality and adapt their tests:

> *We need the changes to the test part because the requirements section in a big corporation, it's challenging to change the requirements or documentation. —P8*

This creates a problematic situation where teams have to work around poor requirements quality by modifying tests instead of fixing the root problem. When organizational processes make it harder to correct requirements than to adapt tests, teams naturally choose the easier path, which allows requirements quality problems to persist and get worse over time. This approach embeds ad-hoc or assumption-based expectations into tests, rather than correcting the root problem. This approach allowing requirements quality problems to persist and intensify over time.

Another major issue was analysed that testers often get involved too late in the process to help improve requirements quality. Participants said that when testing expertise is not included early enough, requirements end up missing important details that are needed for stable tests.

One participant described this problem:

> *When it comes to requirements, they were written a long time ago, and we (as testers) were not involved in their creation. By the time we come into the process, the requirements are already considered final. —P3*

When testing knowledge is excluded from requirements development, important qualities like being testable, measurable, and technically complete details get overlooked. By the time requirements are finalized, they already contain ambiguities, missing conditions, and specifications that can not be properly tested, which sets the stage for flaky tests.

The analysis also revealed that that organizational communication and collaboration processes affect how complete and consistent requirements are. Participants identified systematic gaps in how teams share information, which directly impacts requirements quality, especially in technical details and implementation context that are crucial for stable test development. The lack of standardized processes for documenting requirements emerged as a significant factor that compromises completeness.

One participant explained organizational documentation practices create quality gaps:

> *Flaky tests were often caused by requirement gaps, especially missing low-level technical details. Our teams focused primarily on mid-level requirements, and the lack of standardized documentation across teams resulted in misunderstandings. —P1*

When low-level technical requirements are missing, testers have to guess about implementation details, timing constraints, and system behaviors. The lack of standardization across teams makes this problem worse that how much detail and what

types of information get captured in requirements documents. These completeness gaps directly contribute to test flakiness by leaving critical implementation aspects undefined and open to interpretation.

A participant highlighted this critical information management issue:

> *Critical information is often undocumented and remains only in people's heads, and every individual executes testcase from their perspective if nothing is written down —P4*

This creates several quality problems: critical information is not available to all team members who need it for test design, creating gaps in the documented requirements. Additionally, this personal knowledge can be lost when people leave, making requirements unstable over time, and it means that tests may be designed based on incomplete information, leading to assumptions and interpretations that contribute to unpredictable behavior.

Therefore, this theme captures practitioners' perceptions of the relationship between requirements quality and test flakiness, directly addressing RQ1. Participants emphasized that organizational procedures, constraints, and the late involvement of testers often hinder the maintenance of high-quality requirements, which in turn increases test flakiness. Their accounts indicate that while practitioners recognize the importance of requirements completeness and clarity, systemic barriers frequently prevent this knowledge from being effectively applied, contributing to the occurrence of flaky tests. These insights further contributed to guidelines that emphasized tester involvement, systematic documentation, and flexible change management.

4.2.7 Theme 5: Practitioner Assessment of Requirements-Flakiness Relationship Significance

The analysis reveals that practitioners not only recognize requirements quality as a contributing factor to test flakiness but can also quantify its impact, rank different quality attributes by significance, and understand how contextual factors influence the strength of this relationship. This theme represents practitioners' explicit evaluations of the requirements-flakiness connection, supporting and extending the survey findings that showed diverse but substantial estimates of requirements impact (with most estimating 21-60% of flaky tests stem from requirements issues).

Participants demonstrated a clear understanding of how requirements quality affects test flakiness by providing specific estimates of how much flaky test behavior they attributed to requirements problems, based on experience across different domains. This ability to quantify the impact suggests that their understanding is grounded in long-term observation and analysis.

One participant contrasted two domains:

> *When we were working in the real estate domain, I would say around*

> *15–20% of flaky test issues could be attributed to requirement problems. But once I moved to the banking domain, the process became more structured and formal. As a result, I've seen fewer flaky test issues caused by requirements, maybe just 5–7%. But it's still there to some extent. —P4*

This detailed comparison reveals several important insights. First, practitioners can quantitatively estimate how much requirements contribute to the problem, indicating they have been systematically observing and analyzing this relationship over time. Second, the impact of requirements problems varies significantly by domain, showing that practitioners understand the relationship isn't uniform across contexts. Third, they recognize that organizational process maturity affects how requirements influence flakiness.

When asked directly about the influence of requirements, practitioners expressed strong confidence:

> *I believe requirements plays a major role when it comes to flaky tests —P2*

This straightforward statement indicates that practitioners don't view requirements as merely one small factor among many, they consider them a major cause of flaky tests. The confidence in this statement suggests this assessment is based on substantial experience dealing with these problems rather than speculation.

Another participant explained their thinking about why requirements are so important:

> *Requirements are the foundation of any system, and testing is based on them. If the requirements are unclear or poor in quality, testing becomes ineffective and can, in some cases, lead to flaky tests. —P8*

This reflects a foundational understanding that if tests are based on flawed requirements, they are inherently unstable, leading to inconsistent results the defining characteristic of flaky tests. Not all practitioners agreed that requirements play a major role; others viewed them as a contributing factor rather than the primary cause.An important insight across participants was that the impact of requirements on flakiness depends on context, particularly the maturity of organizational processes. Consequently, this theme directly addresses RQ1 by capturing the nuanced and conditional perceptions of practitioners regarding the relationship between requirements quality and test flakiness, highlighting that its impact varies across domains and organizational contexts. These findings shaped the guidelines by placing requirements practices in their organizational and contextual settings.

5

Discussion

This study examined how industrial practitioners perceive the relationship between requirements quality and test flakiness, and which specific quality attributes contribute to this problem. Through a mixed-methods approach combining survey data from practitioners and semi-structured interviews with participants, this research provides empirical evidence for a relationship that is often discussed informally but has received limited systematic attention in existing literature. The discussion is structured around the three research questions: RQ1 and RQ2 are addressed by interpreting how practitioners perceive the requirements–flakiness relationship and which attributes most significantly contribute to it, while RQ3 is addressed by synthesizing practitioner experiences into concrete guidelines for enhancing requirements quality and mitigating test flakiness.

5.1 RQ1: How do industry practitioners perceive the relationship between requirements quality and the flakiness of test cases?

Awareness Versus Consistent Practice

The study reveals a notable pattern emerged in the relationship between practitioner awareness and behavior regarding requirements quality. While 80% of practitioners recognize that requirements completeness frequently impacts test stability, only 56% consistently consult requirements when investigating flaky tests. This distinction should not be seen as a disregard for documentation, but rather as an indication of how practitioners strategically prioritize and assess potential causes. Flaky tests can arise from a range of sources [37] environmental instability, memory constraints, concurrency issues, or defects in implementation that have no direct link to requirements. In these situations, experienced practitioners understand that reviewing documentation is unlikely to resolve the problem and instead focus on technical diagnostics. This targeted approach demonstrates professional judgment rather than a knowledge practice gap. In contrast, when the observed pattern flakiness indicates a likely requirements related issue, most notably unclear timing behaviors, incomplete specifications, or ambiguous success conditions, practitioners are significantly more inclined to consult the documented requirements or engage stakeholders for

clarification. The survey and interview findings highlight that omissions in detailed specifications and use of ambiguous subjective terms, such as timing related details, in particular, are a recurring contributor to instability, making them a common trigger for revisiting requirements documentation. This behavior demonstrates that practitioners employ context-sensitive reasoning rather than adopting a uniform, indiscriminate approach .

The survey finding that 48% of practitioners seek stakeholder clarification when handling problematic requirements further supports this interpretation. Rather than indicating inconsistent behavior, this demonstrates that practitioners recognize when requirements consultation is likely to be productive and engage accordingly. The interview data reveals that this selectivity is also influenced by practical factors: practitioners may avoid consulting requirements that are outdated, inaccessible, or organizationally protected, focusing instead on actionable investigative approaches.

Domain-Specific Contextual Understanding

The study revealed notable variation in how requirements contribute to test flakiness across domains, reflecting practitioners' sophisticated contextual understanding. For example, participant P5, with experience in both the real estate and banking sectors, reported that approximately 15–20% of flaky tests were attributable to requirements issues in real estate, whereas in the more structured banking domain, this proportion dropped to 5–7%. This comparison is particularly instructive, as it reflects the same practitioner's observations across domains, controlling for individual assessment differences.

Healthcare and banking organizations, operating under strict regulatory requirements and formalized processes, generally maintain more rigorous requirements practices. Interview participant P1 (healthcare) confirmed that, while flaky tests still occur, requirements-related issues contribute relatively little. In contrast, P2 (banking) noted that despite systematic practices, requirements still play a significant role in flaky tests. These perspectives indicate that even within the same domain, the impact of requirements on test flakiness varies across organizations depending on processes, culture, and implementation practices.

Automotive systems practitioners (P3, P4, P6, and P8) provided a distinct perspective, emphasizing that requirements often become outdated, and organizations frequently adjust tests rather than updating specifications. In this domain, incomplete or ambiguous timing specifications were particularly critical, with time-related omissions causing severe flakiness due to deterministic expectations in safety-critical, real-time systems. Although timing-related issues were noted across domains (77% overall), their consequences are amplified in high-criticality environments.

Collectively, these domain-specific observations underscore that, while requirements quality is widely recognized as relevant to test flakiness (RQ1), its specific impact is context-dependent. Organizational processes, regulatory constraints, and system criticality shape how requirements deficiencies manifest in flaky tests. This nuanced

understanding demonstrates that practitioners not only recognize the general relationship between requirements and flakiness but also apply domain-specific reasoning to prioritize interventions and assessments.

Reactive Vs Preventive Approaches

The interview findings reveal that practitioners predominantly employ reactive rather than preventive approaches to requirements related flakiness. Theme 4 (Requirements Quality Governance & Change Management) shows that many practitioners described modifying tests to accommodate unclear and ambiguous requirements, rather than working to correct the requirements themselves. In some cases, updating requirements was perceived as more difficult than adjusting tests, creating a cycle where symptoms are addressed while root causes persist and potentially intensify.

This reactive behavior directly informs RQ1, highlighting practitioners' perception of the relationship between requirements quality and test flakiness. Practitioners recognize that ambiguous, incomplete, or outdated requirements are factors that cause flaky tests. However, organizational constraints and process difficulties often limit their ability to proactively improve requirements, reinforcing a reliance on reactive solutions.

The preference for seeking stakeholder clarification (48% of practitioners) over more systematic preventive measures indicates that flakiness is often perceived as stemming from ambiguity and incompleteness rather than purely technical test issues. This shows that practitioners are aware of the causal role of requirements in test instability, but practical limitations influence how they act on this knowledge. Consequently, the persistence of reactive practices helps explain the awareness practice gap: even when practitioners understand the importance of requirements quality, organizational and process barriers restrict preventive interventions.

Overall, this pattern underscores that addressing requirements-related flakiness requires not just individual awareness but organizational commitment to preventive approaches, demonstrating that practitioners' experiences and perceptions align with the core concern of RQ1, the influence of requirements quality on flaky tests.

5.2 RQ2: Which requirements quality attributes most significantly contribute to test flakiness?

The Impact of Requirements Quality Attributes

Survey responses established a hierarchy of requirements quality issues most often linked to flaky tests: unspecified timeout durations emerge as the most critical factor, followed by unclear success/failure conditions, absent acceptance criteria, and incomplete data specifications. This hierarchy provides important insights into the nature of requirements related flakiness. The predominance of timeout-related issues suggests that temporal specifications represent a critical blind spot in current

requirements practices. Unlike functional requirements, timing constraints are often treated as implementation details rather than explicit requirements and therefore omit important details from specifications.

The interview findings reveal how such omissions trigger a cascade of issues: developers introduce implicit timing assumptions during implementation, while testers form their own interpretations in the absence of explicit specifications. These independent, unaligned assumptions increase the likelihood of mismatches between intended and tested behavior, making tests highly sensitive to environmental variations and resulting in inconsistent, flaky outcomes. The daylight saving time example from participant P4 illustrates how seemingly stable systems can possesses temporal vulnerabilities for years before manifesting as flakiness. The high frequency of unclear success/failure conditions points to a fundamental challenge in translating subjective requirements into objective test criteria. As participant P3, "comfortable" truck braking example demonstrates qualitative language, while these terms are meaningful to human stakeholders, creates significant challenges for automated testing,which requires precise, measurable conditions. This finding suggests that requirements elicitation practices may need to incorporate testing considerations more systematically, ensuring that all requirements are expressed in clear and testable terms. Finally, the grouping of acceptance criteria and data specifications indicates that completeness issues are multifaceted. The interview analysis reveals that these are not independent problems but interconnected aspects of requirements maturity. Missing acceptance criteria often correlate with incomplete data specifications. This pattern indicates that weaknesses in one aspect of requirements quality often reflect broader shortcomings in organisational requirements practices, where maturity across attributes tends to rise or fall together.

Time-Related Specifications as a Critical Blind Spot

The finding that 77% of practitioners experience flakiness due to unspecified timeout durations represents more than just a high-frequency problem, as it reveals a fundamental blind spot in current requirements practices. This issue is particularly seen in automotive domains, where practitioners reported that timing issues and signal timing problems are especially critical due to real time system constraints and hardware-software integration requirements.

Unlike other functional requirements, which have mostly established elicitation and specification techniques, time-related requirements often lack systematic approaches and are left to developer judgment during implementation. In these domains, practitioners emphasised that timing specifications are not simply performance considerations but define critical system behaviours and when left unspecified or ambigious, they introduce variability in how the system responds under different conditions, leading to inconsistent or intermittent behaviour during testing. Such instability manifesting as tests that pass or fail unpredictably creates flakiness in real-time and safety-critical systems, where timing precision is essential.

The interview findings provide crucial insight into why temporal issues are so prob-

lematic across domains. The daylight saving time example from participant P4 illustrates how temporal vulnerabilities can remain hidden for years before manifesting as test flakiness. A long-stable scheduling application began exhibiting inconsistent behaviour following a daylight saving time change. The problem arose because some employees' shifts began during the "missing" hour created by the time adjustment. In some instances, the system processed the shift data correctly; in others, it produced unexpected results or flagged errors, with no clear pattern. This edge case processing logic, which had not been specified in the requirements, created intermittent and difficult-to-reproduce issues that are the indicators of flakiness despite the system having operated reliably for years under normal conditions. This example shows that temporal requirements are not limited to setting timeout values, they also need to account for different edge cases related to timing interactions with external systems and environmental factors. The research reveals timing issues take on an added level of complexity. Signal timing and the precise coordination of inter-component communication are critical, with even microsecond-level differences potentially altering system behavior. When such relationships are left unspecified and unclear in the requirements, small variations in execution can lead to intermittent behaviour and difficult to reproduce outcomes during testing. These issues become even more pronounced in hardware–software integration, where additional timing dependencies rarely documented in traditional requirements and introduce further opportunities for flaky behaviour to emerge.

The interview data also reveals that practitioners often lack systematic approaches for defining and managing time-related specifications across the domains. The participant P8 description of making initial timing assumptions and then reactively adjusting them based on test results illustrates the ad hoc nature of current practices. This reactive approach creates a cycle where tests are adapted to undocumented or poorly understood system behaviours rather than being based on explicit requirements, a pattern that increasing the likelihood of inconsistent behavior.

Ambiguity and Translation into Tests

The finding that practitioners experience flakiness due to unclear success and failure conditions highlights a fundamental challenge in translating stakeholder intent into testable specifications. The interview examples illustrate this challenge vividly: Participant P3 "comfortable" truck braking and participant P2 undefined name format specifications demonstrate how seemingly reasonable requirements specifications becomes untestable when testing and creates uncertainty in the stability of tests. This translation challenge is particularly acute in automating testing environments, where precise, objective criteria are essential for consistent execution. Human testers can apply contextual judgment to ambiguous requirements, making reasonable interpretations based on domain knowledge and stakeholder feedback. Automated tests, however, require explicit, deterministic criteria that can be evaluated consistently across different execution environments and time periods. The interview findings reveal that ambiguity creates tests that pass or fail based on coincidence rather than genuine system behavior. Participant P2's example of tests passing only when randomly generated names "accidentally matched the expected format" represents

a manifestation of requirements-induced flakiness. Such tests create false signals about system quality, passing when they should fail and failing when they should pass, undermining confidence in the entire testing process. The ambiguity problem is rooted in the fact that requirements documents often use qualitative language that is meaningful in human communication but problematic for testing. Terms like "fast," "comfortable," "user-friendly," or "as soon as possible" convey useful information in stakeholder discussions but provide insufficient precision for test automation. This highlights the need for structured approaches to transform qualitative stakeholder requirements into measurable and testable specifications.

Completeness as a Multi-Dimensional Issue

The finding that 61-62% of practitioners experience flakiness due to missing acceptance criteria and incomplete specifications reveals that completeness is a multi-faceted challenge with different dimensions and implications. The interview analysis shows that completeness deficiencies manifest in various ways: missing preconditions (participant P3), omitted critical parameters (participant P6 calculation method), and absent environmental specifications (participant P2 browser compatibility issues). Missing preconditions create particular problems because they force test designers to make assumptions about initial system states. When these assumptions prove incorrect under certain conditions, tests exhibit inconsistent behavior that appears random but reflects unstated environmental dependencies. This type of completeness deficiency is particularly hidden because tests may function correctly during initial development and fail unpredictably in different environments or over time. The omitted critical parameters identified by participant P6 illustrate how seemingly incomplete requirements can hide critical gaps. A requirement that specifies a calculation method but omits key parameters may appear adequate during requirements review but proves insufficient during implementation and testing. This suggests that completeness assessment requires not just a review of what is specified but a systematic analysis of what might be missing. Environmental specification gaps, such as participant P2 browser compatibility issues, reveal that completeness must extend beyond functional requirements to encompass operational context. Tests that pass consistently in development environments may fail unpredictably in production due to unspecified environmental assumptions. This suggests that completeness assessment should systematically address deployment contexts, platform variations, and integration dependencies.

The findings also reveal that ambiguity and incompleteness do not always occur independently; in the mentioned cases above, they interact and compound each other's effects. When these issues overlap, they make it even harder to spot and fix problems in requirements, which means it's important to tackle both ambiguity and incompleteness together.

Interconnected Quality Attribute Relationships

The quality attribute impacts around 61-65% suggests that these issues are not independent problems but interconnected aspects of requirements maturity. Orga-

nizations with poor practices in one area are likely to struggle with others, creating effects that impact flakiness problems. The interconnected nature of these issues impacts both the identification and resolution of problems. From a diagnostic perspective, it has been observed that flaky tests often have multiple contributing requirements quality issues rather than a single root cause. *A test that behaves inconsistently may be affected by unclear success conditions AND missing timeout specifications AND incomplete definitions.* This multiplicity of causes makes root cause analysis more complex and suggests that systematic approaches are needed to identify the full scope of contributing factors. From an intervention perspective, it suggests that making small, isolated fixes might not be very effective. Addressing timeout specifications while leaving success/failure conditions unclear may reduce some flakiness but not eliminate it entirely. This indicates that requirements quality improvement initiatives should take comprehensive approaches that address multiple attributes simultaneously rather than focusing on single issues. The interconnectedness also suggests that organizational factors play a crucial role in requirements quality outcomes. Organizations with mature requirements processes tend to perform well across multiple requirements quality attributes, while those with immature processes struggle broadly. It is suggested that organizational dimension indicates that sustainable improvements require process-level changes rather than just technique-level modifications.

5.3 RQ3: What guidelines can be derived from practitioner experiences to enhance requirements quality and mitigate test flakiness?

Based on the comprehensive analysis of survey and semi-structured interview data, this section addresses RQ3 by presenting guidelines derived from practitioner experiences. These guidelines highlight critical aspects of requirements quality identified through the empirical study and provide actionable strategies for mitigating test flakiness in industrial practice.

5.3.1 Requirements Completeness

Generalized Guidelines

1. **Specify Complete Data Formats and Constraints:** Define exact structure, length, type, allowed characters, and encoding (e.g., UTF-8, Windows-1252). Include regional or special character requirements.
2. **Explicitly Define Temporal and Operational Boundaries:** Specify timeouts, processing delays, daylight saving transitions, leap years, and timezone handling.
3. **Include Validation and Error Handling Rules:** Clearly define acceptable inputs, invalid data, and system responses for each case.
4. **Account for Edge Cases and Special Scenarios:** Consider rare but critical inputs, boundary values, and unusual character sets in testing.
5. **Ensure Data Integrity Constraints:** Specify which data or system components must remain unchanged after operations.
6. **Document Gaps Explicitly:** For incomplete requirements, note missing details, assumptions, and required clarifications.
7. **Validate Requirements:** Verify requirements from the testers' perspective using example test cases.
8. **Document Critical Parameters and Inter-Parameter Relationships:** Include all calculation parameters and their dependencies with other inputs.
9. **Specify System Preconditions and Environmental Context:** Clearly define system initial states, required preconditions, environmental dependencies (hardware, software, network), platform compatibility, integration contexts, external system dependencies, and deployment environments (production, testing, development).

Implementation Example: The following description, shared by participant P2, illustrates how complete requirements should look like:

Participant P2 Description

Complete Specification:
The system must generate random test data in the specified name format.
Format: 3-digit or 5-digit numeric code + alphanumeric string.
Character Set: English letters (A–Z, a–z) and Nordic characters å, ä, ö.
Encoding: Windows-1252.
Validation: Mismatch in format/characters triggers failure.
Data Integrity: Location data remains unchanged.

Incomplete Specification:
The system must generate random test data in the specified name format.
Gaps: Missing format rules, allowed character sets, encoding, error handling, and data integrity constraints.

5.3.2 Requirements Unambiguity

Generalized Guidelines

1. **Replace Subjective Language:** Avoid words like "comfortable," "fast," and "user-friendly." Use quantifiable metrics instead.
2. **Define Measurable Success Criteria:** Establish objective thresholds, numeric limits, or observable conditions.
3. **Specify Failure Scenarios Explicitly:** Clearly define triggers, error states, and expected system responses.
4. **Use Structured Formats (BDD Approach):** Employ the preconditions (Given), actions (When), and expected outcomes (Then) format, as used in Behavior-Driven Development (BDD) [21], to write unambiguous requirements and reduce interpretation variability.
5. **Provide Concrete Examples:** Include both successful and failed scenario examples for clarity.
6. **Decompose Broad Requirements:** Break down high-level requirements into actionable, testable components.
7. **Specify Interaction Flows Step-by-Step:** Document user interactions and system responses explicitly, avoiding assumptions.
8. **Clarify Scope Boundaries:** Define what is included/excluded to prevent interpretation differences.
9. **Explicit Temporal Specifications:** Define timeouts, processing delays, daylight saving adjustments, leap years, timezone handling, and real-time constraints.

Implementation Example:

Unambiguous Specification:
The system must acknowledge user input within 200ms and complete processing within 2 seconds under normal load conditions ($\leq$ 100 concurrent users). Timeout handling must account for daylight saving time transitions."

Ambiguous Specification:
The system should respond quickly to user input.

5.3.3 Other Practical Guidelines

Organizational and technical factors influence requirements accuracy and alignment with system needs

1. **Stakeholder Validation:**
   - Conduct structured requirements reviews with cross-functional teams including testers.
   - Establish regular stakeholder validation throughout the requirements development lifecycle.
   - Document assumptions and verify them with appropriate stakeholders.
2. **Technical Feasibility Validation:**
   - Include technical experts in requirements validation processes.
   - Verify the implementation feasibility for all specified requirements.
   - Identify technical constraints that may affect requirement correctness.
   - Validate integration assumptions with external systems and dependencies.
3. **Real-World Constraint Alignment:**
   - Identify real-world operating conditions that may affect system behavior.
   - Validate requirements against actual usage patterns and environmental conditions.
   - Consider long-term operational scenarios, including edge cases and unusual conditions.
   - Test requirements assumptions in production-like environments.
   - Update requirements based on operational experience and feedback.

These guidelines directly respond to the practitioner concerns identified in RQ1, where 80% recognizing requirements quality impact test flakiness, only 56% consistently apply this knowledge in practice. By providing structured approaches to the specific quality attributes identified in RQ2, these guidelines bridge the awareness-practice gap revealed in the empirical findings. In summary, the proposed guidelines offer concrete, evidence-based approaches for practitioners to systematically reduce requirements-related test flakiness. Derived from survey responses and semi-structured interviews across multiple domains, the guidelines address key aspects of requirements completeness, unambiguity, and practical validation processes. By specifying complete data formats, operational boundaries, measurable success criteria, and structured stakeholder and technical validation practices, organizations can

bridge the gap between recognized requirements issues and consistent implementation. Applying these guidelines can improve test reliability, reduce the proportion of flaky tests attributed to requirements deficiencies, and encourage more predictable and robust system behavior in practice.

5.4 Integration of Mixed-Methods Findings

The integration between survey and interview data strengthens confidence in the findings while revealing important aspects. The survey established the quantitative scope of the problem, while interviews explained the mechanisms behind the patterns. For example, the survey finding that 77% of practitioners experience timeout-related flakiness gains deeper meaning through interview examples showing how temporal edge cases can remain hidden before causing system inconsistencies. The mixed-methods approach also revealed important contextual factors that pure quantitative or qualitative approaches might have missed. The survey captured domain variation in requirements impact, while interviews explained why organizations with formal processes experience lower requirements-related flakiness than organizations with informal processes. This demonstrates that the requirements-flakiness relationship is not universal but mediated by different other contexts. Importantly, the qualitative findings help explain apparent contradictions in the survey data. While practitioners recognize requirements issues as significant contributors to flakiness, their inconsistent consultation of requirements during troubleshooting reflects practical constraints rather than lack of awareness.

5.5 Theoretical Contributions

This study offers several theoretical contributions to the literature on software testing and requirements engineering.

First, it empirically demonstrates that flaky tests are not solely the result of technical deficiencies but are also rooted in the quality of software requirements. While existing literature has focused primarily on code or environment-related causes of flakiness [18, 33], this study demonstrates that these technical manifestations often originate from software requirements deficiencies. By recognizing requirements quality as an early causing factor, this study shifts the understanding of flaky tests from a strictly technical perspective towards a socio-technical framework.

Second, the study extends requirements quality frameworks by identifying specific attributes such as incompletness, ambiguity, and time-related aspects that impact test flakiness. Traditional requirements quality models emphasize correctness, completeness, and consistency [32, 30, 27], but rarely consider their downstream effects on flaky tests.

Third, the findings contribute to understanding the organizational dimensions of software quality. The governance and change management patterns uncovered that requirements quality emerges not only from individual specification practices but

also from team communication, stakeholder alignment, and change control processes. This supports the view of software engineering as a socio-technical system while extending it to the requirements–testing interface.

Finally, the study provides a methodological contribution by demonstrating how a mixed-methods design combining quantitative patterns with qualitative practitioners' insights can uncover multi-layered causes of flakiness. This approach demonstrates how integrating data collection types strengthens theory building in empirical software engineering.

5.6 Threats to Validity

This section discusses threats to validity, focusing on how the research questions were answered, and how this undermines the reliability and validity of the findings.

Internal Validity

The survey sample was mostly composed of practitioners involved in testing (96%), which may have biased the results toward a testing focused view of the relationship between requirements and flaky tests. Testers might be more inclined to attribute flaky tests to issues in requirements rather than to their own testing practices, making the role of requirements appear more significant. Both, survey responses and interview accounts rely on practitioners' retrospective assessments of flaky test incidents and their causes. Practitioners' estimates of how much flakiness stems from requirements issues may be influenced by recent experiences, particularly memorable cases, or cognitive biases that favor certain types of explanations over others. The eight practitioners who participated in interviews were self-selected from the broader survey sample based on their willingness to participate. This introduces potential selection bias, as individuals who volunteer for interviews may hold stronger opinions, have more vivid experiences, or possess different characteristics compared to the wider population. Consequently, this may affect the representativeness and generalizability of the qualitative insights used to complement and explain the quantitative findings. The study captures practitioners' perspectives at a specific point in time, but both requirements practices and testing technologies are rapidly evolving. The relationship between requirements quality and test flakiness may change as organizations adopt new development methodologies or testing technologies (AI-assisted testing).

External Validity

The survey sample, while diverse in experience and industry representation, may not be representative of the broader software development population. The study's recruitment methods may have attracted practitioners who are particularly interested in or experienced with flaky test problems, potentially overestimating the prevalence and impact of requirements related issues. While the study includes participants from multiple industries (software development 55%, automotive 26%, finance 11%), the findings may not generalize uniformly across all software development domains.

Specialized domains such as aerospace, medical devices, gaming, or artificial intelligence systems may have different requirements, practices, and different relationships between requirements quality and testing outcomes. The study does not explicitly address geographic or cultural factors that may influence requirements, practices, and their relationship to testing outcomes. Different regions may have varying approaches to documentation, stakeholder communication, and process formality that could affect the generalizability of findings.

Construct Validity

Both the quantitative and qualitative approaches rely primarily on practitioner perceptions rather than objective measurements of requirements quality or flakiness rates. While these perceptions offer valuable insights into how practitioners experience and understand the relationships between these constructs, they introduce potential threats to construct validity. Practitioners' subjective assessments of requirements quality may be influenced by individual experiences, organizational contexts, or cognitive biases such as the availability heuristic, wherein recent or memorable incidents disproportionately affect their perceptions. The study operationalizes complex constructs, such as "requirements quality attributes" and "test flakiness," through survey items and interview questions. However, these measures may not fully capture the multidimensional nature of these phenomena. In particular, requirements quality encompasses numerous attributes beyond those measured here, and the interplay between these attributes in real-world contexts may be more nuanced than what the selected indicators reveal.

5.7 Future Work

The limitations identified in this study highlight several promising opportunities for future research. First, expanding the qualitative dataset to include a broader and more diverse range of practitioners would enhance the depth, validity, and generalizability of insights regarding how requirements deficiencies influence test behavior. Additionally, empirical validation using real-world software projects is essential to strengthen and contextualize these findings. This could involve analyzing actual requirements documents alongside test execution data, establishing traceability links between requirements and test cases, and applying NLP-based tools for automated, scalable assessment of requirements quality. Such data driven approaches would enable researchers to systematically examine the relationship between specific requirements quality attributes and the occurrence of flaky tests, providing stronger causal evidence and actionable insights for improving software quality.

Furthermore, the organizational aspects of the requirements flakiness relationship requires deeper investigation. Research exploring how different organizational structures, communication patterns, and change management approaches affect requirements quality and its relationship to testing outcomes could inform organizational design decisions.

The domain variation identified in this study also suggests significant value in industry specific research. Detailed investigations into requirements practices across domains such as healthcare, automotive, and financial services could reveal domain specific patterns and facilitate the development of tailored guidance.

6

Conclusion

Test flakiness remains a pervasive challenge in software development, undermining developer confidence, increasing maintenance costs, and producing misleading signals about system quality [36, 54, 48]. While the technical causes of flaky tests have been extensively studied, the impact of requirements quality on test instability has received limited attention.

This mixed-methods study explored the relationship between flaky tests and requirements quality from industrial practitioners' perspectives, addressing three primary research questions: (RQ1) How do practitioners perceive the connection between requirements quality and test flakiness, (RQ2) Which specific requirements quality attributes contribute to flakiness and (RQ3) What guidelines can be derived from practitioner experiences to enhance requirements quality and mitigate test flakiness? By analyzing survey data from 74 practitioners and conducting eight semi-structured interviews, the study provides empirical evidence linking requirements deficiencies to test instability, while also proposing actionable practitioner-informed guidance to reduce flakiness.

The findings demonstrate that the relationship between requirements quality and test flakiness is significant, context-dependent, and informed by sophisticated practitioner reasoning. RQ1 showed that practitioners widely acknowledge requirements as a contributor to test instability, though their awareness is not always systematically applied in practice. RQ2 highlighted the most critical attributes such as completeness, unambiguity, and timing specifications that practitioners associate with flaky tests. RQ3 extended these insights into concrete, evidence-based guidelines that translate practitioner experiences into structured improvement strategies.

The evidence-based guidelines framework bridges research and practice, offering organizations a systematic approach to reducing test instability. These recommendations are grounded in empirical observations rather than theoretical assumptions, making them practical and feasible for real-world organizational contexts and constraints. Most importantly, the study establishes requirements quality as a measurable factor in software reliability, deserving systematic attention alongside traditional technical quality factors.

Ultimately, the contribution of this work lies not only in documenting the require-

ments–flakiness relationship but also in providing both empirical foundations (RQ1, RQ2) and actionable guidelines (RQ3) for practitioners to systematically improve software quality through better requirements practices.

Declaration of Generative AI

During the preparation and writing of this thesis, generative AI tools most notably ChatGPT was used in a limited and clearly defined manner. The tool was employed primarily to assist the author in improving phrasing for academic english and in structuring LaTeX code for formatting tables, figures, and references.

Bibliography

[1] *RE '02: Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, USA, 2002. IEEE Computer Society.

[2] So/iec/ieee draft standard for software and systems engineering–software testing–part 1: Concepts and definitions. *ISO/IEC/IEEE P29119-1/DIS, September 2012*, pages 1–64, 2012.

[3] Iso/iec/ieee international standard - systems and software engineering – life cycle processes – requirements engineering. *ISO/IEC/IEEE 29148:2018(E)*, pages 1–104, 2018.

[4] Azeem Ahmad, Erik Norrestam Held, Ola Leifler, and Kristian Sandahl. Identifying randomness related flaky tests through divergence and execution tracing. In *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 293–300, 2022.

[5] Azeem Ahmad, Ola Leifler, and Kristian Sandahl. Empirical analysis of practitioners' perceptions of test flakiness factors. *Software Testing, Verification and Reliability*, 31(8), August 2021.

[6] Ozlem Albayrak and Duygu Albayrak. The impact of software development companies' on software engineers' responses to incomplete requirements. *International Journal of Information Studies*, 1:272–279, 01 2009.

[7] Anis R. Amna and Geert Poels. Ambiguity in user stories: A systematic literature review. *Information and Software Technology*, 145:106824, 2022.

[8] Issa Atoum, Mahmoud Khalid Baklizi, Izzat Alsmadi, Ahmed Ali Otoom, Taha Alhersh, Jafar Ababneh, Jameel Almalki, and Saeed Masoud Alshahrani. Challenges of software requirements quality assurance and validation: A systematic literature review. *IEEE Access*, 9:137613–137634, 2021.

[9] Khalil Bahia and Mohammed Najm. Quality requirements analysis of utilization in the systems of a financial institution. *International Journal of Computer Applications*, 178(47):1–7, 2019.

[10] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology.

Qualitative Research in Psychology, 3:77–101, 01 2006.

[11] David Carmo, Luísa Gonçalves, Ana Dias, and Nuno Pombo. Improved flaky test detection with black-box approach and test smells. In *2023 IEEE Symposium on Computers and Communications (ISCC)*, pages 245–251, 2023.

[12] Kaushal Chari and Manish Agrawal. Impact of incorrect and new requirements on waterfall software project outcomes. *Empirical Software Engineering*, 23, 02 2018.

[13] Andrei Contan, Catalin Dehelean, and Liviu Miclea. Test automation pyramid from theory to practice. In *2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, pages 1–5, 2018.

[14] Marcello Cordeiro, Denini Silva, Leopoldo Teixeira, Breno Miranda, and Marcelo d'Amorim. Shaker: a tool for detecting more flaky tests faster. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1281–1285, 2021.

[15] John W. Creswell and Vicki L. Plano Clark. *Designing and Conducting Mixed Methods Research*. SAGE Publications, Thousand Oaks, CA, 3rd edition, 2018.

[16] Hafsa Dar. Reducing ambiguity in requirements elicitation via gamification. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*. IEEE, 2020.

[17] P. L. de Souza, A. F. do Prado, W. L. de Souza, S. M. dos Santos Forghieri Pereira, and L. F. Pires. Improving agile software development with domain ontologies. In S. Latifi, editor, *Information Technology - New Generations*, volume 738 of *Advances in Intelligent Systems and Computing*, pages 307–313. Springer, Cham, 2018.

[18] Moritz Eck, Fabio Palomba, Marco Castelluccio, and Alberto Bacchelli. Understanding flaky tests: the developer's perspective. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, page 830–840, New York, NY, USA, 2019. Association for Computing Machinery.

[19] Sakina Fatima, Taher A. Ghaleb, and Lionel Briand. Flakify: A black-box, language model-based predictor for flaky tests. *IEEE Transactions on Software Engineering*, 49(4):1912–1927, 2023.

[20] Daniel Méndez Fernández, Stefan Wagner, Marcos Kalinowski, André Schekelmann, Ahmet Tuzcu, Tayana Uchoa Conte, Rodrigo Oliveira Spínola, and Rafael Prikladnicki. Naming the pain in requirements engineering: Comparing practices in brazil and germany. *IEEE Software*, 32:16–23, 2015.

[21] Martin Fowler. Given when then, August 21 2013. Accessed: 2025-08-16.

[22] Julian Frattini. *Good-Enough Requirements Engineering*. PhD thesis, Blekinge Tekniska Högskola, 2025.

[23] Google Testing Blog. Where do our flaky tests come from?, 2017. Accessed: 2025-05-30.

[24] Martin Gruber and Gordon Fraser. Flapy: Mining flaky python tests at scale. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 127–131, 2023.

[25] Sarra Habchi, Guillaume Haben, Mike Papadakis, Maxime Cordy, and Yves Le Traon. 14a qualitative study on the sources, impacts, and mitigation strategies of flaky tests. In *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 244–255. IEEE, 2022.

[26] Guillaume Haben, Sarra Habchi, Mike Papadakis, and Yves Le Traon. The importance of discerning flaky from fault-triggering test failures: A case study on the chromium ci, February 2023.

[27] Mohd. Haleem, Md. Faizan Farooqui, and Md. Faisal. Tackling requirements uncertainty in software projects: A cognitive approach. *International Journal of Cognitive Computing in Engineering*, 2:180–190, 2021.

[28] Debaro Huyler and Craig M McGill. Research design: Qualitative, quantitative, and mixed methods approaches, by john creswell and j. david creswell. thousand oaks, ca: Sage publication, . *New Horizons in Adult Education & Human Resource Development*, 31(3), 2019.

[29] Massila Kamalrudin. Automated software tool support for checking the inconsistency of requirements. In *2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 693–697, 2009.

[30] Massila Kamalrudin and Safiah Sidek. A review on software requirements validation and consistency management. *International Journal of Software Engineering and Its Applications*, 9:39–58, 10 2015.

[31] Soren Lauesen. Software requirements-styles and techniques. 2002.

[32] Larissa A Lopes, Eduardo G Pinheiro, Tiago S da Silva, and Luciana AM Zaina. Using uxd artefacts to support the writing of user stories: Findings of an empirical study with agile developers. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*, pages 1–4, 2018.

[33] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. An empirical analysis of flaky tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, page 643–653, New York, NY, USA, 2014. Association for Computing Machinery.

[34] Lloyd Montgomery, Davide Fucci, Abir Bouraffa, Lisa Scholz, and Walid Maalej. Empirical research on requirements quality: a systematic mapping study. *Requir. Eng.*, 27(2):183–209, June 2022.

[35] Owain Parry, Michael Hilton, Gregory M. Kapfhammer, and Phil McMinn. What do developer-repaired flaky tests tell us about the effectiveness of automated flaky test detection? In *2022 IEEE/ACM International Conference on Automation of Software Test (AST)*, pages 160–164, 2022.

[36] Shanto Rahman, Aaron Massey, Wing Lam, August Shi, and Jonathan Bell. Automatically reproducing timing-dependent flaky-test failures. In *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 269–280, 2024.

[37] Shawn Rasheed, Amjed Tahir, Jens Dietrich, Negar Hashemi, and Lu Zhang. Test flakiness' causes, detection, impact and responses: A multivocal review, 2022.

[38] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, 14(2):131–164, April 2009.

[39] August Shi, Alex Gyori, Owolabi Legunsen, and Darko Marinov. Detecting assumptions on deterministic implementations of non-deterministic specifications. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 80–90, 2016.

[40] Denini Silva, Martin Gruber, Satyajit Gokhale, Ellen Arteca, Alexi Turcotte, Marcelo d'Amorim, Wing Lam, Stefan Winter, and Jonathan Bell. The effects of computational resources on flaky tests. In *2024 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 101–101, 2024.

[41] Brijendra Singh and Shikha Gautam. The Impact of Software Development Process on Software Quality: A Review. In *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*, India, December 2016. IEEE.

[42] Bosun Sogeke. How poor requirements lead to catastrophic software failures. `https://medium.com/@sogekebosun/how-poor-requirements-lead-to-catastrophic-software-failures-aab25399f398`, January 20 2025. Medium, 4min read.

[43] Ian Sommerville. *Software Engineering*. Pearson Education Limited, Boston, 10 edition, 2016.

[44] George Spanoudakis and Andrea Zisman. Inconsistency management in software engineering: Survey and open research issues. *Handbook of Software En-*

gineering and Knowledge Engineering, 11 2000.

[45] G Stella, R Marsura, A Messina, and S Rizzo. Capturing user needs for agile software development. In *Proceedings of 4th International Conference in Software Engineering for Defence Applications*, volume 422, pages 191–202, 2016.

[46] Klaas-Jan Stol and Brian Fitzgerald. The abc of software engineering research. *ACM Trans. Softw. Eng. Methodol.*, 27(3), September 2018.

[47] Richard Berntsson Svensson and Björn Regnell. Aligning quality requirements and test results with quper's roadmap view for improved high-level decision-making. In *2015 IEEE/ACM 2nd International Workshop on Requirements Engineering and Testing*, pages 1–4, 2015.

[48] Swapna Thorve, Chandani Sreshtha, and Na Meng. An empirical study of flaky tests in android apps. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 534–538, 2018.

[49] Suresh Thummalapenta. Improving developer productivity via flaky test management. `https://devblogs.microsoft.com/engineering-at-microsoft/improving-developer-productivity-via-flaky-test-management/`, February 2022. Accessed: 2025-05-30.

[50] Ruixin Wang, Yang Chen, and Wing Lam. ipflakies: a framework for detecting and fixing python order-dependent flaky tests. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, ICSE '22, page 120–124, New York, NY, USA, 2022. Association for Computing Machinery.

[51] Rotem Wiess, Vered Holzmann, and Moti Frank. The factors affecting the quality of the software requirements specifications in technological projects: A report on a research in progress. *IFAC Proceedings Volumes*, 45(6):1101–1104, 2012.

[52] Mohammad Zakeri-Nasrabadi and Saeed Parsa. Natural language requirements testability measurement based on requirement smells. *Neural Computing and Applications*, 36:13051–13085, 2024.

[53] Yi Zhao, Yun Hu, and Jiayu Gong. Research on international standardization of software quality and software testing. In *2021 IEEE/ACIS 20th International Fall Conference on Computer and Information Science (ICIS Fall)*, pages 56–62, 2021.

[54] Celal Ziftci and Diego Cavalcanti. De-flake your tests: Automatically locating root causes of flaky tests in code at google. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 736–745. IEEE, 2020.

A

Appendix

A.1 Survey Questions

Demographics

1. Which phase of the Software Development Life Cycle (SDLC) is your work most closely related to?
   (a) Requirements
   (b) Design
   (c) Development
   (d) Testing
   (e) Deployment & Maintenance
2. How many years of experience do you have in your field?
   (a) 0–3 years
   (b) 4–6 years
   (c) 7–10 years
   (d) More than 10 years
3. How do you define the nature of your work domain?
   (a) Tech & Software Development
   (b) Automotive
   (c) Finance & Banking
   (d) E-commerce & Healthcare

(e) Others

Flaky Test and Requirement Quality

4. How often have you encountered flaky tests (test cases that sometimes pass and sometimes fail without changes to the code under test)?

   (a) Always

   (b) Frequently

   (c) Occasionally

   (d) Rarely

   (e) Never

5. Do you refer back to requirements/user stories/features list while investigating flaky tests?

   (a) Always

   (b) Frequently

   (c) Occasionally

   (d) Rarely

   (e) Never

6. How do you typically handle requirements that you believe might lead to flaky tests?

   (a) Request clarification from stakeholders before writing tests

   (b) Write tests with additional error handling and fallback mechanisms

   (c) Document assumptions and implicit behaviors in test cases

   (d) Create multiple test scenarios to cover different interpretations

   (e) Others

7. How likely are the following issues in the requirement specification to contribute to flaky tests? (Missing acceptance criteria, unclear success/failure conditions, unspecified timeout durations, incomplete data specifications)

   (a) Always

   (b) Frequently

(c) Occasionally

(d) Rarely

(e) Never

8. Based on your experience or knowledge, do you think the completeness of requirements impacts the stability of tests?

   (a) Always

   (b) Frequently

   (c) Occasionally

   (d) Rarely

   (e) Never

9. In your experience, how often have you encountered flaky tests caused by poorly defined test conditions such as preconditions, inputs, or expected outcomes?

   (a) Always

   (b) Frequently

   (c) Occasionally

   (d) Rarely

   (e) Never

10. In your experience/knowledge, approximately what percentage of flaky tests stem from requirements issues versus implementation issues?

    (a) 0–20% (Almost None)

    (b) 21–40% (Small Proportion)

    (c) 41–60% (About Half)

    (d) 61–80% (A Large Proportion)

    (e) 81–100% (All)

A.2 Interview Guide

Demographics

1. How many years of experience do you have in the industry?
2. Can you briefly describe your role?

Flaky Tests and Requirements

3. Can you share a specific case where you encountered flaky tests that you believe were caused by requirement quality issue?
   (a) What behavior or feature were these tests trying to validate?
4. Requirement quality is commonly assessed using attributes such as unambiguity, completeness, correctness, and consistency.
   (a) How have any of these attributes posed challenges to flaky tests in your experience?
   (b) What was the specific issue in the requirement?
   (c) How was it eventually resolved?
   (d) Were any changes made to the requirements document as a result?
5. How do you think the requirements (or lack of good requirements) contributed to the flakiness of those tests?
6. From your experience, what strategies have helped reduce flakiness caused by requirement-related issues?
7. Is there anything else you would like to share about the relationship between requirements and flaky tests that was not covered in this interview?