Federal University of São Carlos– UFSCar
Center for Exact Sciences and Technology– CCET
Department of Computing– DC
Graduate Program in Computer Science– PPGCC

Camilo Hernán Villota Ibarra

Towards a strategy and tool support for test generation based on good software testing practices: classification and prioritization

São Carlos
2025

Camilo Hernán Villota Ibarra

Towards a strategy and tool support for test generation based on good software testing practices: classification and prioritization

Ph.D. Thesis presented to the Graduate Program in Computer Science of the Center for Exact Sciences and Technology at the Federal University of São Carlos, as part of the requirements for obtaining the title of Doctor in Computer Science.

Area of concentration: Methodologies and Techniques of Computing

Advisor: Prof. Dr. Auri Marcelo Rizzo Vincenzi
Co-advisor: Prof. Dr. José Carlos Maldonado

São Carlos
2025

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Tese de Doutorado do candidato Camilo Hernan Villota Ibarra, realizada em 13/05/2025.

Comissão Julgadora:

Prof. Dr. Auri Marcelo Rizzo Vincenzi (UFSCar)

Prof. Dr. Daniel Lucrédio (UFSCar)

Prof. Dr. André Takeshi Endo (UFSCar)

Prof. Dr. Eduardo Santana de Almeida (UFBA)

Prof. Dr. Plínio Roberto Souza Vilela (UNICAMP)

O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

I thank God for always being with me, and my family for believing in me and supporting me unconditionally.

Acknowledgements

I thank God, with all my heart, for always being by my side, guiding my steps, and providing everything necessary to get here. He gave me strength in times of doubt, courage in times of challenge, and an unwavering faith to continue this path. Without Him, nothing would have been possible.

I am very grateful to my parents, who, with their love and dedication, always allowed me to fly. I truly honor them. To my girlfriend, who motivated me along the way with love. To my brother, who is always willing to help.

I thank all those who have supported me in some way or another. I will always be grateful for their generosity and trust in me.

I express my gratitude to the Federal University of São Carlos and the institutions that sponsored me along this path. Their support was crucial in making this achievement a reality.

To all the professors, colleagues, and administrative staff who were part of this process, thank you for your support, your experience, and for contributing to the growth of this work.

I would especially like to thank Professor Dr. Auri Marcelo Rizzo Vincenzi and Professor Dr. José Carlos Maldonado, who not only offered me their knowledge, support, and motivation, but, above all, their friendship. The opportunity to learn from you as professors and, above all, to experience your example as human beings is something I will never forget. Thank you for believing in me and for being a constant source of inspiration along this path.

God has done it once again...

"Be strong and courageous. Do not be afraid; do not be discouraged, for the Lord your God will be with you wherever you go."
(Joshua 1:9)

Summary

This thesis addresses the challenge of defining, validating, and operationalizing good software testing practices, with a focus on improving the quality of test cases. It begins with a systematic literature review, which identifies 131 practices from 103 primary studies, refined into a set of 40 essential best practices. These practices were classified into code-oriented and no-code-oriented groups and validated through a practitioner-centered survey involving experienced software testers, who evaluated their clarity, relevance, applicability, and prioritization. Based on this foundation, the thesis introduces TAI-EvalGenTCS. This AI-based tool uses the OpenAI GPT-4 turbo model to evaluate test cases against proposed practices and generate optimized test cases. The tool was experimentally validated using 16 real-world Java projects, demonstrating its effectiveness in improving the modularity, maintainability, and compliance of the test cases with best practices. The survey results provided empirical confirmation of the proposed classification and offered valuable insights into tool support needs and industry perceptions. The contributions of this research include a structured and validated set of testing practices, a ranked list of essential test case features, empirical validation from industry practitioners, and a practical AI-powered solution that bridges the gap between test automation and test design quality.

Keywords: Good software testing practices, test cases, test case development practices, test case generation tools.

Resumo

Esta tese aborda o desafio de definir, validar e operacionalizar boas práticas de testes de software, com foco na melhoria da qualidade dos casos de teste. Inicialmente, foi realizada uma revisão sistemática da literatura, que identificou 131 práticas a partir de 103 estudos primários, posteriormente refinadas em um conjunto de 40 práticas essenciais. Essas práticas foram classificadas em grupos orientados a código e não orientados a código e validadas por meio de uma pesquisa centrada em profissionais experientes da área de testes de software, que avaliaram sua clareza, relevância, aplicabilidade e priorização. Com base nessa fundamentação, a tese apresenta a TAI-EvalGenTCS, uma ferramenta baseada em inteligência artificial que utiliza o modelo OpenAI GPT-4 turbo para avaliar casos de teste em relação às práticas propostas e gerar casos de teste otimizados. A ferramenta foi experimentalmente validada utilizando 16 projetos reais em Java, demonstrando sua eficácia na melhoria da modularidade, manutenibilidade e conformidade dos casos de teste com as boas práticas. Os resultados da pesquisa forneceram confirmação empírica da classificação proposta e informações valiosas sobre necessidades de suporte por ferramentas e percepções da indústria. As contribuições desta pesquisa incluem um conjunto estruturado e validado de práticas de teste, uma lista hierarquizada das principais características dos casos de teste, validação empírica junto a profissionais da indústria e uma solução prática apoiada por IA que conecta a automação de testes à qualidade do design de casos de teste.

Palavras-chave: Boas práticas de testes de software, casos de teste, práticas de desenvolvimento de casos de teste, ferramentas de geração de casos de teste.

List of Publications

The following is a list of publications made during the doctoral program up to the present.

❏ Full article in conference (under evaluation):

- **Best Practices, Essential Features, and Tool Support for Developing Good Test Cases.** Camilo Hernán Villota Ibarra, Pedro Henrique Kuroishia, José Carlos Maldonado, Auri Marcelo Rizzo Vincenzi. Authors: Camilo Hernán Villota Ibarra (Federal University of São Carlos), Pedro Henrique Kuroishia (University of São Paulo), José Carlos Maldonado (University of São Paulo), Auri Marcelo Rizzo Vincenzi (Federal University of São Carlos).
- **TAI-EvalGenTCS: An AI-Based Tool for Evaluation and Generation of Test Case Sets Based on Best Practices.** Camilo Hernán Villota Ibarra, José Carlos Maldonado, Auri Marcelo Rizzo Vincenzi. Authors: Camilo Hernán Villota Ibarra (Federal University of São Carlos), José Carlos Maldonado (University of São Paulo), Auri Marcelo Rizzo Vincenzi (Federal University of São Carlos).

❏ Accepted article:

- **Mock Objects: A Case Study in Industry.** Camilo Hernán Villota Ibarra, Daniel Lippi Castro de Faria, André Takeshi Endo, Delano Medeiros Beder, Auri Marcelo Rizzo Vincenzi. Published in: *Proceedings of the XIX Brazilian Symposium on Information Systems, 2023.* DOI: <https://doi.org/10.1145/3592813.3592930>

List of Figures

List of Tables

Contents

Chapter 1

Introduction

This chapter presents the context of the research, the motivation behind it, the research hypothesis, and the objectives of this research.

1.1 Context

Software testing practices are essential to ensure the delivery of high-quality software products (DOBSLAW; FELDT; NETO, 2022). The integration of various testing methodologies, tools, and frameworks is crucial for effective testing, particularly in the context of rapid software development demands (IHME et al., 2014). A test case is a fundamental component in software testing, serving as a detailed set of conditions, data inputs, and expected outcomes to validate software functionality (ALMOG; HEART, 2009). Test cases are crucial for verifying software systems' design, functionality, and development aspects (TAIPALE; SMOLANDER, 2006; WIJAYASIRIWARDHANE; WIJAYARATHNA; KARUNARATHNA, 2011). Good software testing practices involve prioritizing test cases, ensuring their quality, and using efficient techniques to detect faults (RAO et al., 2022a; BARRAOOD; MOHD; BAHAROM, 2022; KOCHHAR; XIA; LO, 2019). Different methods and tools can be used for test case generation (AMRITA; GUPTA, 2021; FETA; FITRIA, 2022; MEIXNER et al., 2020; NAHEED; NADEEM; ZAMAN, 2022). These tools are used in the software industry to generate and manage test cases to ensure the reliability of software products (VERMA; CHOUDHARY; TIWARI, 2023; LI; GALL; SPASESKI, 2017; ROLANDO; VITA, 2016).

Good test cases are essential for detecting faults in a software process and improving software quality (CATAL, 2012). However, there is a lack of consensus on what constitutes a good test case. Some studies have evaluated hypotheses regarding test case quality but found no evidence to support them (LUCRéDIO et al., 2023). In addition, the criteria for what makes a test case good are still unclear (BARRAOOD; AL., 2021). Best practices

in software testing can be defined as actions or principles recognized for their positive results and replicated (GRANERO-GALLEGOS; PHAN; NGU, 2023). This ambiguity surrounding test case quality poses significant challenges in achieving effective software testing practices. This thesis focuses on practices, framed in the definition of good test cases, with the objective of improving the definition of good software testing practices.

To define what constitutes a good test case, a characterization and classification process was conducted (see Chapter 2), narrowing down a set of 40 refined best practices from an initial pool of 131 identified in this research (see Chapter 3). These practices serve as practices for software testers and are categorized according to their focus. Code-oriented practices focus on the technical and structured aspects of test case design, ensuring precision, reliability, and maintainability. These are further divided into common sense practices, based on experience and fundamental principles, and literature-supported practices, grounded in research and academic findings. On the other hand, no-code-oriented practices address broader testing activities, including test automation and management, which optimize efficiency through tools and structured processes, and business-oriented practices, which ensure that testing strategies align with organizational goals.

Practices are ranked based on the frequency of references, the quality assessments from primary studies, and the highest H-index of authors contributing to those studies.

This thesis explores tools that support test case generation and management, acknowledging their role in improving testing efficiency and overall software quality.

In addition, this study presents the development of a tool named TAI-EvalGenTCS, which leverages artificial intelligence to support the evaluation and generation of test cases (see Chapter 4). The tool uses the defined best practices as its evaluation criteria and employs OpenAI's GPT-4-turbo model to analyze a given test case, score its compliance, and produce an improved version that aligns with good testing principles. By automating this evaluation and refinement process, the tool contributes to improving the maintainability, structure, and quality of test cases in practice.

Moreover, to ensure the relevance and acceptance of the defined practices, a survey was conducted with experienced software testing practitioners (see Chapter 5). This allowed the validation of the proposed classification and offered insights into how professionals perceive, adopt, and prioritize these practices in real-world testing contexts.

In summary, this research makes the following contributions:

- ❑ **A systematic characterization study:** This study conducts a structured review of software testing practices, analyzing 103 primary studies selected from an initial pool of 1,763 studies. The first phase, which focused on extracting practices from the literature, identified 131 distinct testing practices proposed by these studies. In the second phase, a thorough analysis was conducted to evaluate the applicability and relevance of each practice. As a result, this refinement process led to a final selection of 40 best practices derived from a subset of 15 primary studies, which

provided the most substantial and practice-oriented insights. These 15 studies were the foundation for the structured set of best practices, ensuring their validity and practical applicability in test case development.

- **A refined and structured set of best practices:** The final 40 practices are categorized into two major groups based on their focus:
  - **Code-Oriented:** These focus on the technical structure of test cases to ensure effectiveness and maintainability.
    - **Common Sense:** Practices derived from experience, intuition, and practical understanding.
    - **Literature-Supported:** Recommendations grounded in academic research and empirical studies.
  - **No-Code-Oriented:** These address higher-level testing strategies that go beyond code implementation.
    - **Test Automation and Management:** Techniques leveraging automation and structured management for increased efficiency.
    - **Business-Oriented:** Practices ensuring that testing objectives align with organizational goals and priorities.
- **Survey validation:** A survey was conducted with industry practitioners to validate the classification, understand the relevance and applicability of the practices, and gain insight into current challenges faced in test case design and evaluation.
- **Development of an AI-based tool (TAI-EvalGenTCS):** A practical tool was designed and implemented to support test case evaluation and generation. It applies the structured best practices and provides automated feedback and suggestions for improvement, enhancing the quality and maintainability of test cases.
- **Evaluation of tools:** The study examines tools for test case generation, management, and automation, emphasizing their role in improving testing efficiency, coverage, and reliability across different software development contexts.

1.2 Motivation

The quality of software testing directly influences the reliability and maintainability of software systems (JIA, 2023). Despite advances in test automation, the quality of individual test cases is often overlooked (BAQAR; KHANDA, 2024; ALEKSANDROVA; DOBRYNINA, 2024). Poorly designed test cases lead to ineffective testing, increased debugging efforts, and ultimately software defects that persist into production (PONTILLO, 2024; DU, 2023).

Furthermore, the software industry lacks a shared and validated set of best practices for what constitutes a high-quality test case. Although several studies propose guidelines, these are often dispersed, inconsistent, or lacking validation through practitioner feedback.

Therefore, this thesis is motivated by the need to:

- ❏ Define and validate what makes a test case "good" based on structured and empirical practices.
- ❏ Provide practitioners with a clear, categorized, and actionable set of best practices for test case development.
- ❏ Offer a tool that evaluates and improves test cases according to these practices, bridging the gap between automation and quality.

Moreover, to ensure that the proposed best practices are not only theoretically grounded but also practically applicable, an additional empirical validation was conducted through a practitioner-centered survey. This survey gathered insights from experienced professionals and provided critical evidence on the clarity, applicability, and prioritization of the selected practices.

1.3 Research Gap and Rationale

Several empirical and practitioner-based studies have investigated aspects of test case quality and best practices (CATAL, 2012; LUCRéDIO et al., 2024; KOCHHAR; XIA; LO, 2019; DIAS-NETO et al., 2017; BAQAR; KHANDA, 2024). These works highlight relevant insights into prioritization, practitioner perceptions, regional practices, and the impact of test smells. However, they remain fragmented, often lack systematic validation, and provide limited consensus on what constitutes a "good" test case.

This lack of consolidation reinforces the need for a structured and transparent synthesis of existing evidence. For this reason, this thesis conducts a Systematic Literature Review (Chapter 2) to integrate prior findings, classify and refine best practices, and establish a robust foundation for further validation and tool development.

1.4 Hypothesis

This thesis is based on the hypothesis that a systematic and validated set of good software testing practices can support the evaluation and generation of high-quality test cases, and that an AI-based tool can effectively apply these practices to improve test case maintainability, structure, and compliance with testing goals.

1.5 Objectives

The general objective of this work is to contribute to the definition, validation, and application of good software testing practices based on practitioners and to operationalize them through an AI-powered tool for evaluating and generating test cases.

The specific objectives are:

- ❑ **Identify** good software testing practices reported in the literature and professional practice.
- ❑ **Classify** these practices into relevant categories (code-oriented and no-code-oriented).
- ❑ **Refine** a set of essential practices for designing high-quality test cases.
- ❑ **Validate** the relevance and applicability of these practices through a practitioner-centered survey.
- ❑ **Develop** a support tool (TAI-EvalGenTCS) that operationalizes these practices.

1.6 Thesis Structure

This thesis is organized into chapters, each of which addresses a fundamental aspect of the research and collectively contributes to the definition, validation, and operationalization of good software testing practices.

- ❑ **Chapter 2 – Literature Review:** This chapter presents the systematic literature review conducted to explore the current state of research on good software testing practices. It details the search strategy, inclusion and exclusion criteria, and data extraction procedures. The findings from 103 primary studies serve as the basis for identifying and synthesizing recurring testing practices and gaps in existing tools (see Chapter 2).
- ❑ **Chapter 3 – Good Software Testing Practices:** This chapter presents the theoretical foundation of the thesis. It introduces a systematic characterization and classification of good software testing practices derived from the literature. A total of 131 practices were initially extracted from 103 primary studies. Through a multi-stage analysis process, this number was refined to a set of 40 best practices, which were then categorized into code-oriented and no-code-oriented groups. This chapter also includes the ranking of practices, identification of essential features of good test cases, and a review of tools supporting test case generation and management (see Chapter 3).

- **Chapter 4 – TAI-EvalGenTCS: An AI-Based Tool for Evaluation and Generation of Test Case Sets Based on Good Software Testing Practices:** This chapter presents the design, implementation, and validation of TAI-EvalGenTCS, a computational tool developed to evaluate and generate test cases based on the validated best practices. The tool leverages OpenAI's GPT-4-turbo model to assess test case quality and automatically produce optimized versions. The architecture, functionalities, and interface of the tool are described in detail. Additionally, an experimental study using 16 Java projects is presented to evaluate the tool's effectiveness in improving test case quality (see Chapter 4).

- **Chapter 5 – Validating Good Software Testing Practices: A Practitioner-Centered Survey:** This chapter presents the design, implementation, and results of a survey conducted with experienced software testing practitioners. The survey aimed to validate the clarity, practical relevance, and prioritization of the 40 best practices proposed in this research. It also examines the perceived usefulness and tool support associated with these practices, offering an essential empirical perspective to the theoretical contributions (see Chapter 5).

- **Chapter 6 – Conclusions, Future Work, and Lessons Learned:** This chapter summarizes the main findings and contributions of the research. It revisits the research hypothesis and objectives, highlights the practical and scientific outcomes of the work, and proposes directions for future research. It also presents reflections and lessons learned throughout the research journey, including insights into software testing practices, test case evaluation, and tool development (see Chapter 6).

1.7 Research Questions and Methodological Approach

This research is guided by the same research questions defined to conduct the Systematic Literature Review (SLR) presented in Chapter 2, which are: (i) What are the best practices for developing good test cases? (ii) What features are considered essential in good test cases? and (iii) How are tools used in the testing process to support good test cases? These questions provided the foundation for the literature review and were extended throughout the study to validate the practices with practitioners and to implement an AI-based tool capable of applying these practices in practice. To answer them, this thesis combines three methodological components: a Systematic Literature Review to identify and classify best practices; a practitioner-centered survey to validate their clarity, relevance, and applicability empirically; and the development and experimental validation of TAI-EvalGenTCS, an AI-based tool that operationalizes the proposed practices to evaluate and improve test cases. Together, these elements ensure that the research questions are fully addressed in both theoretical and practical dimensions.

Chapter 2

Systematic Literature Review

This chapter presents a Systematic Literature Review (SLR) aiming at identifying the state-of-the-art on best practices to write good test cases.

2.1 Contextualization and Background

The purpose of this section is to provide the necessary context for understanding the motivation and scope of the systematic literature review (SLR). Rather than giving a general introduction to the thesis, this section explains why a structured review of existing evidence is necessary. It highlights the key concepts of software testing and test cases, as well as the lack of consensus regarding what constitutes a "good" test case. This contextualization justifies the need for a systematic approach to identify, consolidate, and refine practices that have been proposed across both academic research and industrial practice.

2.1.1 Software Testing

Software testing is a critical phase in the software development lifecycle, aimed at verifying that a system meets its specified requirements and identifying potential defects before the system is released. It encompasses a variety of processes, including validation and verification activities, and serves both preventive and corrective purposes. Effective testing ensures software quality, reliability, and usability, reducing maintenance costs and customer complaints (TAIPALE; SMOLANDER, 2006; KOCHHAR; XIA; LO, 2019).

Various testing approaches exist, including manual and automated testing, as well as black-box and white-box techniques, and testing at different levels, such as unit, integra-

tion, system, and acceptance testing. The choice of technique depends on project context, available tools, and specific testing goals (VERMA; CHOUDHARY; TIWARI, 2023; LI; GALL; SPASESKI, 2017).

Although software testing is widely recognized as a critical activity for ensuring software quality (TAIPALE; SMOLANDER, 2006; KOCHHAR; XIA; LO, 2019), different schools of thought emphasize various priorities. Some authors highlight the preventive nature of testing, focusing on early fault detection to reduce maintenance costs (CATAL, 2012). In contrast, others argue that the effectiveness of testing strongly depends on the organizational context and the maturity of the development process (DIAS-NETO et al., 2017). This diversity of perspectives illustrates the persistent challenge of aligning testing processes with evolving industry demands and technological shifts, reinforcing the need for more structured and validated guidance on how testing activities should be planned and executed.

2.1.2 Test Case

A test case is a fundamental unit in software testing. It defines a specific condition or set of conditions under which a tester determines whether a software system behaves as expected. Typically, a test case includes an identifier, a description, input data, execution steps, expected results, and actual results (ALMOG; HEART, 2009). It serves as both a documentation artifact and an execution guide for testers.

Good test cases are reusable, clear, maintainable, and designed to cover both common and edge scenarios. They play a crucial role in measuring software correctness and in the early identification of bugs, reducing the risk of failures in production environments (CATAL, 2012; KOCHHAR; XIA; LO, 2019).

There is also no unified agreement on what constitutes a “good” test case in practice. For instance, (ALMOG; HEART, 2009) define a test case mainly as a verification artifact, while (CATAL, 2012; KOCHHAR; XIA; LO, 2019) extend this view by framing test cases as reusable assets with clear structure, traceability, and maintainability. However, studies like (BARRAOOD; MOHD; BAHAROM, 2022) point out that even with clear documentation, test cases often suffer from redundancy and lack of independence, which affects long-term project costs. These contrasting perspectives highlight the relevance of addressing not only the structural definition of a test case but also its quality attributes and practical implementation challenges.

2.1.3 Good Software Testing Practices

Good software testing practices encompass strategies, techniques, and guidelines that enhance the effectiveness, efficiency, and reliability of the testing process. They encompass principles such as test planning, early test involvement, prioritization of test cases, au-

tomation, continuous feedback, and traceability between requirements and tests (RAO et al., 2022a; BARRAOOD; MOHD; BAHAROM, 2022; GRANERO-GALLEGOS; PHAN; NGU, 2023).

Despite the broad acknowledgment of their importance, there is no unified agreement in the industry or academia about what specific practices define a "good" test case. Practices are often derived from empirical evidence, expert experience, or institutional standards, making their evaluation and systematization a research challenge (LUCRéDIO et al., 2023; BARRAOOD; AL., 2021).

2.1.4 Test Case Quality

Test case quality refers to the extent to which a test case can reliably detect defects and validate correct behavior. Quality is influenced by factors such as clarity, completeness, consistency, independence, and relevance. High-quality test cases are those that maximize fault detection while minimizing maintenance effort (KOCHHAR; XIA; LO, 2019; BARRAOOD; MOHD; BAHAROM, 2022).

Several studies have attempted to define or evaluate test case quality, but findings are often inconclusive or contradictory, highlighting the need for a more standardized and evidence-based approach (LUCRéDIO et al., 2023).

2.2 Research Questions and Objectives

To develop and guide the systematic review of the literature, the following research objectives were defined:

- ❏ What are the best practices for developing good test cases?
  - **Objective**: Identify the best practices experts recommend for developing good test cases.
- ❏ What features are considered essential in good test cases?
  - **Objective**: Determine the specific features experts agree are essential for good test cases.
- ❏ How are tools used in the testing process to support good test cases?
  - **Objective**: Explore how tools are used in the testing process to support the creation of test cases.

2.3 Study selection process

The SLR follows the steps to organize the information in a systematic literature review using the guidelines established by Kitchenham e Charters (2007), a methodology for conducting systematic reviews in software engineering. This protocol provides a structured approach to improve transparency, reproducibility, and methodological rigor in literature reviews. To implement these guidelines effectively, the StArt tool (FABBRI et al., 2016; ZAMBONI et al., 2010), which facilitates the organization and management of systematic reviews, was used. Through StArt, the study objective was defined and recorded, the inclusion and exclusion criteria were established, and the selection process was documented to maintain consistency. Additionally, StArt allowed for the classification and organization of selected studies, streamlining data extraction and analysis. The integration of Kitchenham's structured methodology with StArt's management capabilities enhances the reliability and efficiency of the systematic review process. Following, we described each step of the SLR.

2.3.1 Automatic research

A comprehensive search strategy encompassed significant databases, including IEEE Xplore, Wiley, ACM Digital Library, Web of Science, and Engineering Village. For this, the search string is defined as described below:

The search string is designed to identify relevant primary studies on the quality and best practices of test cases in software testing. It includes various terms related to test cases, such as "test case," "test suite," and "test case generation," ensuring both singular and plural forms are captured. It also incorporates terms related to best practices and quality factors, using wildcards like "best practice*," "good practice", and "good qualit*" to include all relevant variations. Additionally, the search string covers professional and industry contexts with terms like "practitioner," "professional," and "industry." Wildcards broaden the search to capture multiple forms of key terms, ensuring a comprehensive and relevant set of search results.

2.3.2 Removal of duplicate studies

In some instances, identical studies were identified in multiple databases, requiring the removal of duplicates. Although the Start tool (FABBRI et al., 2016; ZAMBONI et al., 2010) was used to detect duplicate studies automatically, it was ineffective in some situations. Consequently, the primary studies were manually analyzed to eliminate any remaining duplicates. To discern such cases, the title, authors, year of publication, and abstract were analyzed, with the least recent publication retained in cases of duplication.

2.3.3 Inclusion and exclusion criteria

Studies included in this review encompass a range of sources, including academic journals, conference proceedings, and industry reports, with a focus on software testing practices and tools. The study inclusion criteria are described below:

- ❏ The study reports models that define or represent what is a good test case.
- ❏ To produce pools of recent references, the year of publication was kept within the range of 2010 to 2024.
- ❏ The study defines features of good test cases.
- ❏ The study reports a support tool that allows you to judge the quality of a set of test cases.
- ❏ The study reports a support tool that allows generating a set of test cases respecting the features of good test cases.
- ❏ The study reports models that define or represent what is a good test case.

In addition, the study exclusion criteria are described below

- ❏ The written language is not English.
- ❏ The study does not address issues mentioned in any inclusion criteria.
- ❏ In case of duplicate studies, the most complete one is considered.
- ❏ The primary study is a summary, a short course description, tutorial, lecture, copyright form, or overview of an event (for example, a conference or workshop).
- ❏ The study is not available for download.
- ❏ The study is published in gray literature.

2.3.4 Data extraction

The online search produced 1,763 primary studies. Table 1 presents the number of primary studies retrieved from each online database. Subsequently, the next phase involves removing duplicate studies and applying inclusion and exclusion criteria to the collected primary studies.

The initial screening involved reviewing titles, keywords, and abstracts to filter out primary studies irrelevant to this research. As a result, 103 primary studies passed the initial screening after considering the inclusion and exclusion criteria based on the analysis of keywords, abstracts, introductions, and conclusions.

A second screening involved thoroughly examining the full text of the studies and gathering all relevant information to address the research questions. During this phase, the practices proposed in each primary study were extracted. Of the 103 primary studies initially selected, 131 practices were identified.

The entire process was conducted using the StArt tool (FABBRI et al., 2016; ZAMBONI et al., 2010).

Table 1 – Summary of findings

Source	Number of primary studies
IEEE	331
Wiley	300
ACM Digital Library	283
Web of Science	448
Engineering Village	401
Total	**1,763**

Figure 1 illustrates the process of narrowing down and extracting practices from an initial pool of studies.

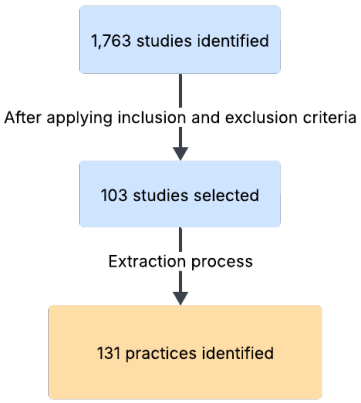

Figure 1 – Extracted of practices

2.3.5 Description of the Analysis Process

To ensure methodological rigour and traceability in the identification and refinement of good software testing practices, a structured process was conducted using a detailed analytical spreadsheet[1]. This spreadsheet served as the central instrument for consolidating, categorizing, and interpreting information from the 103 primary studies initially included in this research. It was designed to systematically organize data from multiple dimensions, enabling a comprehensive evaluation of the practices reported across diverse contexts and study designs.

[1] The complete analysis can be accessed at: <https://docs.google.com/spreadsheets/d/1ugtOYAtjuLByG4Ob5S1FNYGGS5N-nUBdh7JUpVBIyIw/edit?gid=0>

The spreadsheet is structured in multiple sheets, each corresponding to a specific focus area (e.g., surveys of industry practices, reviews of testing approaches, practitioner interviews, and region-specific analyses). Within each sheet, every study was represented as a row and described through a consistent set of analytical dimensions:

- ❏ Title and Reference Link: Facilitated traceability to the original study.
- ❏ Abstract: Summarized the scope and objectives of the study.
- ❏ Contribution: Highlighted the explicit scientific or practical advances reported.
- ❏ Results: Captured key findings relevant to testing practices.
- ❏ Conclusions: Synthesized the study's final insights.
- ❏ Core Concepts: Identified technical or methodological constructs addressed.
- ❏ Application Domain: Specified the context (e.g., industry, academia, regional case studies).
- ❏ Scale or Project Size: Indicated whether practices were validated in small, medium, or large-scale projects.
- ❏ Testing Level: Mapped practices to levels such as unit, integration, or system testing.
- ❏ Automation: Documented references to test automation, tools, or techniques.

This multi-dimensional schema allowed for a granular comparison of studies, enabling not only the extraction of candidate practices but also their contextualization according to factors such as project scale, domain applicability, and testing maturity.

A critical component of this process involved cross-referencing findings across studies to identify recurring patterns. Columns dedicated to Core Concepts and Results were instrumental in detecting overlapping recommendations (e.g., modular test design, comprehensive coverage), while those focused on Automation and Testing Level helped differentiate context-specific practices from those of broader applicability. This analysis provided a robust foundation for distinguishing between practices with widespread relevance and those limited to niche contexts.

Furthermore, the spreadsheet facilitated the systematic filtering and refinement of practices. This included:

- ❏ Grouping redundant or closely related practices, such as consolidating recommendations for edge-case and normal flow testing into broader guidelines for comprehensive input coverage.

- ❏ Excluding practices with insufficient empirical support, identified through cross-referencing low-quality or isolated evidence.
- ❏ Tagging context-dependent practices based on their association with specific domains or project scales.
- ❏ Flagging automation-specific insights to separate tool-oriented recommendations from conceptual testing principles.

2.3.6 Traceability of Results

The synthesis of results was conducted through an iterative and transparent process, fully documented in a structured spreadsheet. Rather than treating the extraction and refinement of practices as isolated steps, the spreadsheet functioned as a methodological backbone that ensured consistency and traceability throughout the review. Each practice was linked to its original primary study, maintaining a direct connection between evidence and the resulting classification.

The spreadsheet captured multiple dimensions of analysis, including:

- ❏ **Study metadata:** titles, abstracts, and reference links to guarantee traceability back to the original sources.
- ❏ **Analytical dimensions:** contributions, results, conclusions, core concepts, application domain, project scale, testing level, and references to automation.
- ❏ **Decision criteria:** whether practices were excluded, merged, or retained, with explicit justifications documented in each case.
- ❏ **Bibliometric indicators:** integration of H-index values and citation counts of the authors, providing additional perspective on the academic weight of the supporting studies.

2.3.7 Spreadsheet-based Analytical Process

To ensure transparency and reproducibility, the data extraction and synthesis process was systematically documented in a structured spreadsheet. Each sheet was designed to capture a specific analytical task:

- ❏ **Results**: Consolidated the set of 103 primary studies, including metadata such as authors, year, venue, and contribution type.
- ❏ **Practices**: The initial extraction of 131 raw practices directly reported in the literature, linked to their sources and context of application.

- ❑ **Relevance**: The assessment of the relevance of the practice in terms of relevance to the design of the test case, clarity, and applicability was recorded.
- ❑ **Quality & Metrics**: The quality assessment of primary studies (empirical support, methodological rigor, clarity) was stored together with bibliometric indicators such as citation counts and H-index.
- ❑ **Ranking**: Integrated all previous evidence to produce a final prioritization of practices.

This spreadsheet-driven process ensured that the transition from raw extraction to refined practices was fully traceable and transparent. Each inclusion, exclusion, or refinement decision was explicitly documented, creating a transparent audit trail that strengthens methodological reliability.

2.3.8 Quality assessment

Verify the quality of the primary studies identified in the process using the quality assessment protocol defined by Dybå and Dingsøyr (DYBÅ; YR, 2008), which is widely adopted for evaluating the academic relevance of studies. These criteria ensure a systematic approach to assessing the quality of systematic studies across various disciplines, including software engineering.

When a given attribute was partially available, it was marked with the value Yes (1). The sum of each attribute determines the final score for this study, which can range from 0 to 11. These references provide a structured approach to define the assessment criteria, ensuring that the questions effectively measure the quality of the study and that the scores align with the systematic evaluation standards.

QC1: Does the study explicitly present itself as a research paper?

QC2: Are the research aims clearly defined and justified?

QC3: Is the study context adequately described to understand its scope?

QC4: Is the research design appropriate for addressing the research objectives?

QC5: Are the study subjects or cases precisely defined and described?

QC6: Does the study include a control group for comparison, if applicable?

QC7: Is the data collection method aligned with the research questions?

QC8: Is the data analysis performed rigorously and systematically?

QC9: Are potential threats to the validity of the study identified and discussed?

QC10: Are the study findings clearly stated and well-supported by the data?

QC11: Does the study provide valuable contributions to research or practice?

This scoring system allows us to reject studies that do not meet the quality criteria defined in the study protocol and to guide the synthesis activity of secondary studies. Primary studies that scored 0 (zero) on all quality criteria described in QC1 were excluded.

To enhance the quality assessment process, the H-index was evaluated to assess the quality of the researchers involved in the primary studies. The H-index measures the highest number of publications with matching citations, providing an intuitive evaluation of influence and productivity (KREINOVICH; KOSHELEVA; NGUYEN, 2020). Table 2 presents the final quality score and the highest H-index for each selected primary study.

2.4 Results of the Review

The systematic literature review yielded 103 primary studies after applying inclusion and exclusion criteria. From these, a total of 131 distinct practices were extracted. A second-level screening identified 15 studies that provided sufficiently detailed and practice-oriented contributions. These 15 studies formed the empirical foundation for the derivation of 40 refined best practices.

2.4.1 Overview of Extracted Practices

The practices identified in the selected studies were highly diverse, ranging from code-level guidelines to organisational and process-oriented strategies. Broadly, the extracted practices can be grouped into the following categories:

- ❑ **Code-Oriented Practices:** Focused on technical aspects of test case design, such as clarity, independence, maintainability, and coverage.
- ❑ **Process-Oriented Practices:** Emphasised planning, prioritisation, traceability, and continuous evaluation.
- ❑ **Tool-Supported Practices:** Addressed the use of tools for test case generation, management, and quality assessment.

This categorisation anticipates the refined taxonomy that is formally presented in Chapter 3.

2.4.2 Key Contributions from the Literature

Several empirical and practitioner-based studies contributed substantially to the understanding of what constitutes a "good" test case:

- (CATAL, 2012) highlighted the importance of prioritisation and the preventive role of testing, stressing that early identification of defects reduces long-term costs.
- (LUCRéDIO et al., 2024) analysed test smells and their negative impact on maintainability, reinforcing the need for practices that minimise redundancy and increase independence.
- (KOCHHAR; XIA; LO, 2019) conducted an extensive practitioner-based study, revealing that clarity, traceability, and automation are consistently valued by professionals in real-world contexts.
- (DIAS-NETO et al., 2017) provided a regional perspective, characterising testing practices in South America and demonstrating how organisational maturity affects the adoption of best practices.
- (BAQAR; KHANDA, 2024) explored emerging challenges and opportunities in test case quality, emphasising the relevance of AI and automation to support evaluation and generation activities.

Together, these studies reveal both convergences and divergences in how good practices are conceptualised. While there is consensus on attributes such as clarity, maintainability, and independence, differences emerge regarding prioritisation, tool support, and contextual applicability.

2.4.3 Synthesis of Findings

The synthesis of the reviewed studies allows us to identify recurring patterns and gaps:

- **Consensus:** Attributes such as clarity, reusability, traceability, and independence appear consistently across multiple studies as hallmarks of good test cases.
- **Fragmentation:** Practices are scattered across different contexts, industries, and research traditions, often lacking systematic validation or a common terminology.
- **Tool Support:** Although tools for test case generation and management are frequently cited, their integration with quality-focused practices remains limited.
- **Research Gap:** Few studies provide consolidated frameworks or validated taxonomies of best practices. This reinforces the need for structured synthesis, as addressed in this thesis.

They narrowed the 131 initial practices to 40 refined best practices, based on three criteria: (i) frequency of citation across primary studies, (ii) quality assessment score of the source study, and (iii) the H-index of the authors involved. These 40 practices form the structured foundation for the classification and ranking presented in Chapter 3.

2.5 Final Considerations

This chapter provided a structured synthesis of the current knowledge on good software testing practices, highlighting convergences, divergences, and research gaps. The SLR not only clarified the fragmented state of the literature but also consolidated the empirical foundation for the derivation of 40 refined best practices. These results serve as the empirical backbone of the next chapter, where the practices are analysed, classified, and systematically presented.

Table 2 – Quality Scores and H-Index of Selected Primary Studies

Name	QC1	QC2	QC3	QC4	QC5	QC6	QC7	QC8	QC9	QC10	QC11	Score	H-Index
Practitioners' Views on Good Software Testing Practices (KOCHHAR; XIA; LO, 2019)	1	1	1	1	1	1	1	1	1	1	1	11	98
Toward the characterization of software testing practices in South America: looking at Brazil and Uruguay (DIAS-NETO et al., 2017)	1	1	1	1	1	1	1	1	1	1	0	10	47
Improving Software Testing by Observing Practice (TAIPALE; SMOLANDER, 2006)	1	1	1	1	1	1	1	1	0	1	0	9	41
Review Paper on Various Software Testing Techniques & Strategies (ANWAR; KAR, 2019)	1	1	1	1	1	1	1	0	1	1	0	9	1
Best Practices for the Organizational Implementation of Software Testing (MAJCHRZAK, 2010)	1	1	1	1	1	1	0	1	1	1	0	9	27
How Good Are My Tests? (BOWES et al., 2017)	1	1	1	0	1	1	1	1	0	1	1	9	44
An initial investigation of the effect of quality factors on Agile test case quality through experts' review (BARRAOOD; MOHD; BAHAROM, 2022)	1	0	1	1	1	0	1	1	0	1	1	8	15
What Makes Agile Test Artifacts Useful? An Activity-Based Quality Model from a Practitioners' Perspective (FISCHBACH et al., 2020)	1	1	0	1	1	1	0	1	1	0	1	8	33
Challenges and industry practices for managing software variability in small and medium-sized enterprises (IHME et al., 2014)	1	1	1	1	0	0	1	1	1	1	0	8	25
Automatically generating test cases for specification mining (DALLMEIER et al., 2012)	1	1	0	1	1	1	0	1	1	1	0	8	69
On introducing automatic test case generation in practice: A success story and lessons learned (BRUNETTO et al., 2021a)	1	1	1	0	1	1	0	1	1	0	1	8	49
Characterization of software testing practices: A replicated survey in Costa Rica (QUESADA-LóPEZ; HERNANDEZ-AGüERO; JENKINS, 2019)	1	0	1	1	0	1	1	0	1	0	1	7	13
Traceability recovery between bug reports and test cases Mozilla Firefox case study (GADELHA; RAMALHO; MASSONI, 2021)	1	1	0	1	1	1	0	0	1	1	0	7	19
A perception of the practice of software security and performance verification (RIBEIRO; CRUZES; TRAVASSOS, 2018a)	1	1	1	0	1	0	1	1	0	1	1	8	47
Test case understandability model (SETIANI; FERDIANA; HARTANTO, 2020)	1	1	1	1	0	1	0	1	1	0	1	8	24

Chapter 3

Good Software Testing Practices

This chapter presents the identification, analysis, and categorization of good software testing practices based on a systematic literature review. It defines a refined set of practices to guide the development of high-quality test cases. Additionally, it explores their semantic structure and classification into code-oriented and no-code-oriented practices. The chapter also addresses the use of tools to support test case generation, highlighting practical solutions, challenges, and insights reported in the literature.

3.1 Refinement and Analysis of Extracted Practices

3.1.1 Analysis of Extracted Practices

Once the 131 extracted practices were identified through this structured spreadsheet-driven process, a second phase was conducted to analyze, evaluate, and refine them based on their applicability and relevance to test case development. This involved a detailed assessment of each practice, considering its context, feasibility, empirical support, and alignment with the approaches proposed by the authors for developing test cases.

Through this refinement, 40 essential best practices were consolidated, representing the most actionable and effective guidelines for test case development.

To streamline the set:

- ❏ 30 practices were eliminated due to redundancy, sharing similar principles or objectives.
- ❏ 25 practices were excluded due to a lack of relevance or specificity to test case development.
- ❏ 20 practices were excluded due to insufficient evidence in the literature.

- ❏ 16 practices were deemed excessively complex or impractical for real-world application.

Figure 2 illustrates the distribution of excluded practices across these refinement factors, providing a comprehensive overview of the decision-making process.

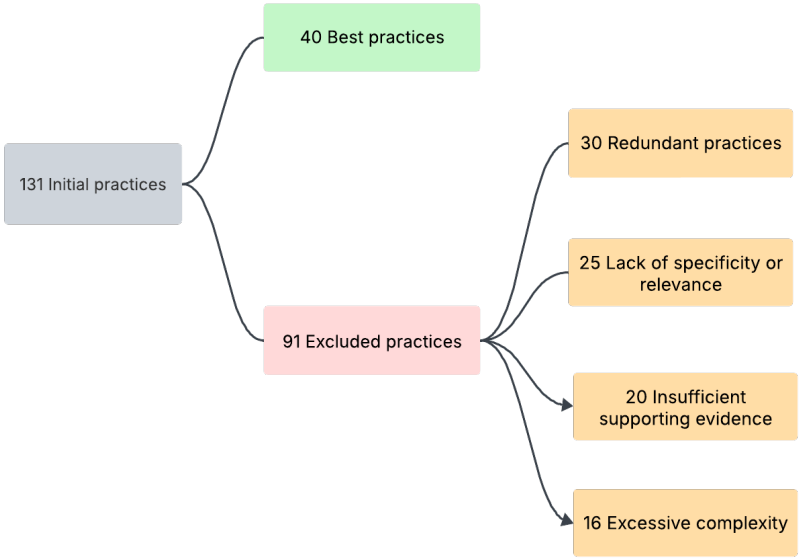

Figure 2 – Distribution of practices excluded during the refinement phase

A deeper investigation into the source studies revealed that, among the 103 primary studies analyzed in the spreadsheet, only 15 studies provided the foundational insights from which these 40 practices were ultimately derived. These studies directly contributed to shaping the final set of best practices, ensuring that the resulting guidelines are both empirically grounded and practically applicable.

This multi-stage, evidence-driven approach—from spreadsheet-based extraction and analysis to refinement and categorization—was essential for maintaining methodological rigor and transparency. It established a clear link between the empirical foundation of the research and the actionable best practices presented in this chapter. Figure 3 illustrates this analysis process, connecting the initial study selection to the extraction, filtering, and final refinement phases.

3.1.2 Semantic Analysis

At this point, a semantic analysis was carried out to analyze and understand the meaning and interpretation of each practice more deeply and contextually. By consolidating knowledge, the aim was to identify patterns, connections, or common themes that might emerge among the various practices. As a result, 40 refined practices were derived from the initial set of 131 practices. Additionally, to improve the understanding of the

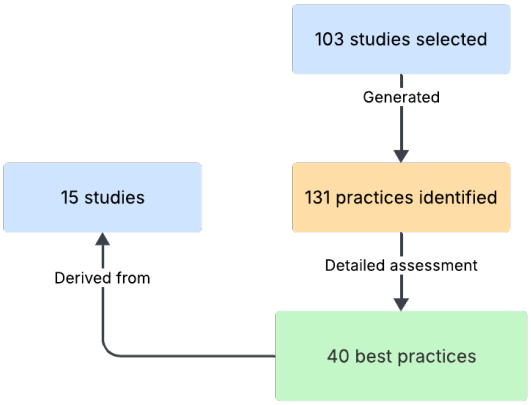

Figure 3 – Analysis of extracted practices

proposed set of best practices, the denomination of each practice is refined by including a nominalized verb, which indicates what is done with the practice, an adjective to demonstrate how it is done, and a noun, which means the object about which it is done (IBARRA et al., 2020; BARóN, 2019).

3.2 Categorization of Practices

At this point, the practices collected are grouped considering their main target: code-oriented or no-code-oriented:

Below are the practices with their respective description:

3.2.1 Code-oriented:

Code-oriented practices focus on the technical and detailed aspects of test case creation and execution. These practices ensure the reliability, accuracy, and maintainability of test cases by directly addressing interactions with code, system functionality, and software design. Code-oriented practices typically require a deep understanding of the software being tested. They provide a basis for designing test cases that are efficient, specific, and technically sound.

Common sense:

This category encompasses practices based on common sense, experience, and a basic understanding of system requirements. It focuses on practical and realistic approaches to test case development.

- ❑ **CS-01 Atomic Specification of Test Cases:** A good test case should be specific or atomic, focusing on one requirement aspect. Test cases within a suite must be

independent, ensuring accurate and reliable results. Independence avoids false positives or negatives and allows for better test coverage by concentrating on specific software aspects (KAMDE; NANDAVADEKAR; PAWAR, 2006; CONG; SHAOYING; ZHENHUA, 2013; DALLMEIER et al., 2012).

- ❑ **CS-02 Complete Independence of Test Cases:** Test cases should be independent to provide flexibility in scheduling and reduce maintenance time and cost (KAMDE; NANDAVADEKAR; PAWAR, 2006). Techniques like cloning support independence through adaptability and reusability (BRUNETTO et al., 2021a; FISCHBACH et al., 2020). An overview of dependencies aids in understanding which test cases to execute based on feature changes (BARRAOOD; MOHD; BAHAROM, 2022; QI; ZHOU, 2022; CHANG; QI, 2017; KANG; LARSEN, 2021).

- ❑ **CS-03 Coverage of Normal and Exceptional Flows:** Good test cases should cover both standard and exceptional flow to thoroughly test the Application's functionality (KAMDE; NANDAVADEKAR; PAWAR, 2006). This approach identifies potential bugs or faults in the system (BRUNETTO et al., 2021a; BARRAOOD; MOHD; BAHAROM, 2022; ANWAR; KAR, 2019).

- ❑ **CS-04 Boundary Values Analysis:** Test cases must perform boundary value analysis by obtaining input values at the edges of an input domain, identifying potential faults at the boundaries (BRUNETTO et al., 2021a; BARRAOOD; MOHD; BAHAROM, 2022; RAMM, 2023; DOBSLAW; FELDT; NETO, 2022).

- ❑ **CS-05 Complete Modularity of Test Cases:** Test cases should be well-modularized, with reference documentation. Well-written, well-commented test code following a consistent style is highly valued (KAMDE; NANDAVADEKAR; PAWAR, 2006; BRUNETTO et al., 2021a; KOCHHAR; XIA; LO, 2019).

- ❑ **CS-06 Detailed Analysis of Size and Complexity:** Most test cases should be small to avoid difficulties in understanding and maintenance. More minor test cases enhance focus, conciseness, and reusability, leading to more efficient testing processes (KAMDE; NANDAVADEKAR; PAWAR, 2006; BRUNETTO et al., 2021a; FISCHBACH et al., 2020; BARRAOOD; MOHD; BAHAROM, 2022; DEBBARMA et al., 2012).

- ❑ **CS-07 Complex Design for fault Detection:** While significant test cases may be necessary to detect problematic faults, a balance is crucial. Increased complexity in a test case can lead to difficulties in understanding and maintaining (FISCHBACH et al., 2020; KOCHHAR; XIA; LO, 2019).

- ❑ **CS-08 Complete Maintenance of Test Code:** Test code must be designed with maintainability, prioritizing good design principles such as modularization and separation of concerns (BARRAOOD; MOHD; BAHAROM, 2022; QUESADA-LóPEZ; HERNANDEZ-AGüERO; JENKINS, 2019).

- ❑ **CS-09 Complete Traceability of Test Cases:** Traceability links between test code, requirements, and source code should be maintained for improved team efficiency and bug detection (KOCHHAR; XIA; LO, 2019; GADELHA; RAMALHO; MASSONI, 2021).

- ❑ **CS-10 Strict Use of Performance and Security Testing:** Different tests, including performance and security tests, are essential for evaluating system performance and identifying security vulnerabilities (BARRAOOD; MOHD; BAHAROM, 2022; QUESADA-LóPEZ; HERNANDEZ-AGüERO; JENKINS, 2019; RIBEIRO; CRUZES; TRAVASSOS, 2018b).

- ❑ **CS-11 Regular Review of Test Cases:** Test cases should be regularly reviewed and updated significantly when requirements change (KAMDE; NANDAVADEKAR; PAWAR, 2006; DIAS-NETO et al., 2017; BOWES et al., 2017).

- ❑ **CS-12 Clear Understanding of Test Cases:** Test cases should be clear, unambiguous, and simple to understand. Involving the test designer in formulating acceptance criteria can enhance the clarity of test cases (KAMDE; NANDAVADEKAR; PAWAR, 2006; DIAS-NETO et al., 2017; BARRAOOD; MOHD; BAHAROM, 2022; SETIANI; FERDIANA; HARTANTO, 2020; FONSECA, 2023).

- ❑ **CS-13 Structured Coverage of Testing Process:** Different integration testing approaches like top-down and bottom-up should be applied based on specific requirements under test (KAMDE; NANDAVADEKAR; PAWAR, 2006; FISCHBACH et al., 2020; HOSSAIN et al., 2023; PATEL; BHATT, 2018; QUDDUS; SINDHU, 2022; AUNG; WIN, 2023).

- ❑ **CS-14 Complete Assurance of Test Code Quality:** Metrics like code coverage are crucial for quality assurance of unit tests (FISCHBACH et al., 2020; BARRAOOD; MOHD; BAHAROM, 2022; KOCHHAR; XIA; LO, 2019).

Literature Supported:

Practices in this category are based on research, studies, and Literature within software testing.

- ❏ **LS-01 Proper Utilization of Test Code Coverage:** Although test code coverage is an important metric in assessing the thoroughness of testing, it alone does not guarantee the effectiveness of the tests. High-quality testing also depends on factors such as clear documentation, efficient test case management, maintainability, and reusability (DIAS-NETO et al., 2017; JASUJA; SINGH, 2017).

- ❏ **LS-02 Required Utilization of Missing Tests:** Utilizing code coverage to identify gaps in test coverage and guide additional test creation is vital. Code coverage metrics offer insights into untested code areas, helping prioritize creating new tests to enhance coverage (DIAS-NETO et al., 2017; UTTING et al., 2020).

- ❏ **LS-03 Efficient Utilization of Test Code Coverage:** A higher test code coverage is not inherently synonymous with better fault detection (DIAS-NETO et al., 2017). While maximizing coverage is an essential metric in evaluating the comprehensiveness of tests, it does not ensure the discovery of all faults. Effective fault detection requires not only extensive coverage but also a strategic focus on specific fault patterns and edge cases, complementing the code coverage with comprehensive testing practices (HUANG et al., 2019; WASEEM et al., 2021; BOWES et al., 2017).

- ❏ **LS-04 Small Test Code Generation Footprint:** Designing test cases with a small execution footprint is essential for efficient and focused testing. By minimizing the amount of code executed in each test case, complexity is reduced, making it easier to identify and isolate issues (KAMDE; NANDAVADEKAR; PAWAR, 2006; DIAS-NETO et al., 2017).

- ❏ **LS-05 Complete Prioritization of Test Cases Design:** Prioritizing the design of test cases to cover diverse requirements is more critical than focusing solely on code coverage. This approach ensures the system meets desired functionality, usability, and performance criteria beyond code-centric considerations (DIAS-NETO et al., 2017; KAMDE; NANDAVADEKAR; PAWAR, 2006; TIUTIN; VESCAN, 2022; RAO et al., 2022b).

- ❏ **LS-06 Responsible Addition of Test Code Maintenance:** Incorporating test cases that address fixed bugs during maintenance is essential to prevent regressions. Regular updates and maintenance of the test suite ensure that the tests remain relevant and effective, ultimately enhancing software quality, reliability, and performance (KAMDE; NANDAVADEKAR; PAWAR, 2006; DIAS-NETO et al., 2017; BRUNETTO et al., 2021a).

- ❑ **LS-07 Suitable Utilization of Test Assertions:** Test assertions are invaluable for detecting subtle faults by explicitly stating expected behavior and comparing it during execution. This formal analysis enhances the fault-detecting ability of testing methods (DIAS-NETO et al., 2017; BRUNETTO et al., 2021a).

- ❑ **LS-08 Responsible Addition of Test Debugging Comments:** Adding comments that document common faults and their possible causes within the test code significantly aids in debugging test failures (HUANG et al., 2019). These comments act as an invaluable form of documentation, providing context and insight for future debugging efforts, thereby streamlining both fault resolution and ongoing test code maintenance (DIAS-NETO et al., 2017; FISCHBACH et al., 2020; HAILPERN; SANTHANAM, 2001)

- ❑ **LS-09 Deterministic Design of Test Results:** Test cases should be designed to produce deterministic results, ensuring reliability and avoiding flakiness. Well-structured test cases with precise, unambiguous steps and clearly defined acceptance criteria lead to consistent and predictable outcomes (KAMDE; NANDAVADEKAR; PAWAR, 2006; DIAS-NETO et al., 2017; HUANG et al., 2019).

- ❑ **LS-10 Complete Avoidance of Test Side Effects:** Test cases must be designed to avoid any side effects to maintain the reliability and repeatability of the testing process. This involves ensuring that tests are independent of each other and do not rely on or modify shared resources, which could lead to unpredictable results (DIAS-NETO et al., 2017; KAMDE; NANDAVADEKAR; PAWAR, 2006).

- ❑ **LS-11 Suitable Utilization of Test Labels and Categories:** Utilizing labels or categories in test cases enables the selective execution of specific test sets, improving the organization and efficiency of the testing process. This approach allows testers to prioritize and run tests based on criteria such as test type, priority, or execution time, facilitating better test management and resource allocation (DIAS-NETO et al., 2017; KAMDE; NANDAVADEKAR; PAWAR, 2006).

3.2.2 No-code-oriented:

At this stage, practices were classified by analyzing each in terms of the criteria, techniques, and levels to which each practice belongs. As a result, the 40 best practices are categorized into 27 code-oriented and 13 no-code-oriented practices. Among the code-oriented practices, 14 fall under the common sense (CS) category, while 11 are Literature supported (LS). The no-code-oriented practices include 9 focused on business orientation (BO) and 6 related to test automation and management (AM).

Business Oriented:

These practices are designed to align closely with administrative goals and requirements. They focus on identifying and prioritizing test scenarios that significantly impact organizational outcomes.

- ❑ **BO-01 Aligned Adjustment of Organizational Testing:** Testing activities should be meticulously aligned with and adjusted according to the business orientation of the Organizational Unit.

  Aligning testing activities with the business orientation of the Organizational Unit is crucial. The nature of the business and its specific requirements significantly influence the testing process and employed strategies. For instance, in software testing, different testing levels and techniques, such as functional, structural, and fault-based testing, should be planned based on specific business needs. Considerations should extend to perspectives like performance, usability, security, and functionality (DIAS-NETO et al., 2017). Additionally, when introducing automated system testing techniques in a business-oriented organization, scalability, reporting, and oracles become critical factors to consider (HUANG et al., 2019; BRUNETTO et al., 2021a; IHME et al., 2014).

- ❑ **BO-02 Responsible Utilization of Risk-based Testing:** Prioritizing testing efforts based on the level of risk associated with different aspects of the system is essential.

  Risk-based testing is an approach that allocates testing efforts according to the level of risk associated with various aspects of the system. This approach ensures a more efficient use of resources by focusing on areas posing the highest risk to the system's quality and functionality. Organizations benefit from identifying critical areas that require thorough testing and allocating resources accordingly. This helps ensure comprehensive testing of essential aspects, reducing the likelihood of potential failures or issues. Risk-based testing improves test coverage and more effective defect identification, resulting in higher-quality software products (HUANG et al., 2019; FELDERER; SCHIEFERDECKER, 2014; IHME et al., 2014).

- ❑ **BO-03 Conscious Impact on Testing Organization:** An organization's business orientation influences the structure and organization of testing processes.

  The business orientation of an organization has a direct impact on the structure and organization of testing processes. Different business orientations may necessitate distinct testing approaches and strategies. For example, a business delivering high-quality software products may have a dedicated testing team and a well-defined testing process. Organizations investing in testing tools and adopting standardized

testing methodologies may exhibit a more structured and organized testing process (DEAK; STåLHANE, 2013; TAIPALE; KARHU; SMOLANDER, 2007; BOWES et al., 2017).

❑ **BO-04 Conscious Impact on Knowledge Management Strategy:** The business orientation significantly influences the strategy for managing knowledge within the testing context.

An organization's business orientation substantially affects the strategy for managing knowledge within the testing context. Business-oriented organizations require testing techniques that can effectively address large industrial applications, including automatic system testing techniques scalable to complex applications and capable of generating easily interpretable test reports (BRUNETTO et al., 2021a). Additionally, domain-specific approaches can be utilized to enhance the effectiveness of testing tools in specific domains, such as exploiting the organization of the GUI in functional areas to make automatic system testing more effective in business-oriented applications (WASEEM et al., 2021).

❑ **BO-05 Carefully Measure Outsourcing Impact:** Business orientation and knowledge management strategy have implications for outsourcing in testing, and the codification of knowledge facilitates testing outsourcing.

Business orientation and knowledge management strategy have implications for outsourcing testing. In the context of testing outsourcing, the codification of knowledge can streamline the process. Organizations can transfer this knowledge to external testing providers more efficiently by documenting and standardizing testing practices. This ensures the alignment of testing activities with the organization's business orientation and requirements. Codification also facilitates effective communication and collaboration between the organization and the outsourcing partner, as both parties clearly understand the testing processes and expectations (CATAL, 2012; HUANG et al., 2019).

❑ **BO-06 Complete Utilization of Testing Methodology:** A defined method or process should be employed for testing activities.

Using methodology or process in software testing entails employing a defined method or process for conducting testing activities. This includes following a structured approach to planning and designing software testing before coding, evaluating the quality of test artifacts, and analyzing identified defects (DIAS-NETO et al., 2017). It also involves identifying and utilizing risks for planning and executing software tests and monitoring adherence to the test process (DEAK; STåLHANE, 2013; T., 2022).

- ❑ **BO-07 Clear Definition of Testing Responsible:** Clearly defining a reliable professional or team for overseeing and executing testing activities is essential.

  A responsible professional or team should be clearly defined to oversee and execute testing activities. This ensures accountability and practical testing. In small companies, software engineers often take on the responsibility of testing at all levels, from unit testing to system testing (DEAK; STåLHANE, 2013). A designated individual or team with a deep understanding of the testing process is crucial to ensuring efficient and effective testing (FISCHBACH et al., 2020; HAMILTON, 2020).

- ❑ **BO-08 Separation of Testing and Development Activities:** The decision to separate or integrate testing and development activities depends on project context and requirements (KAMDE; NANDAVADEKAR; PAWAR, 2006; DIAS-NETO et al., 2017; BRUNETTO et al., 2021a; GéLVEZ, 2011).

- ❑ **BO-09 Early Planning of Testing Plan:** Early involvement of testing in the planning phase is emphasized for better quality software and reduced costs (KAMDE; NANDAVADEKAR; PAWAR, 2006; DIAS-NETO et al., 2017; BRUNETTO et al., 2021a; FISCHBACH et al., 2020; NINDEL-EDWARDS; STEINKE, 2007).

Test Automation and Management:

This category includes practices related to test automation and efficient test case management. It emphasizes the use of tools and automation techniques to improve efficiency.

- ❑ **AM-01 Responsible Utilization of Test Case Automatic Generation:** Utilize automatic generation of test cases responsibly to enhance efficiency while maintaining quality.

  Using tools to generate test cases is highly recommended as they significantly enhance testing efficiency and effectiveness by saving time and effort. Automated test case generation tools have been extensively studied and applied in various testing environments, including unit testing, REST API testing, and system testing (DIAS-NETO et al., 2017; KAMDE; NANDAVADEKAR; PAWAR, 2006). These tools prioritize test cases based on code or interaction coverage criteria, ensuring critical functionalities are tested first (BRUNETTO et al., 2021a). Integrating these tools improves software quality and reliability (HUANG et al., 2019; WINZINGER; WIRTZ, 2022; SUN; WEI; DONG, 2018).

- ❑ **AM-02 Responsible Utilization of Management Tools:** Employing test management tools for systematically monitoring and recording test results is essential.

Test management tools are commonly used for systematically tracking and recording test results. These tools provide features that facilitate the management and organization of test cases, including cloning, adding, moving, and deleting test cases and steps. Test management software includes exporting tests to standard formats, multi-user capabilities, progress tracking for test writing and execution, result recording, linking to requirements, creating coverage matrices, and offering flexible security options (KAMDE; NANDAVADEKAR; PAWAR, 2006). Using test management tools for tracking and recording results is essential in maintaining a structured and organized approach to testing (DIAS-NETO et al., 2017). Integrating these tools into an organization's testing process provides quickly browsable outputs that guide test strategy definition and facilitate test effort planning (BRUNETTO et al., 2021a; BLACK, 2002).

- ❑ **AM-03 Suitable Utilization of Estimation Tools:** The application of tools to evaluate the effort and schedule required for testing activities is crucial.

  Utilizing tools to estimate test effort and schedule is crucial for practical testing activities. These tools assist in evaluating the effort and time required for testing, aiding in efficient planning and resource allocation (ISLAM et al., 2016). Organizations can make informed decisions about resource allocation and scheduling testing activities using such tools. They provide valuable insights into the complexity of the testing process, helping identify potential bottlenecks or areas that require additional attention. Employing tools to estimate test effort and/or schedule is essential for effective test planning and resource management (QUESADA-LóPEZ; HERNANDEZ-AGüERO; JENKINS, 2019; DIAS-NETO et al., 2017).

- ❑ **AM-04 Suitable Utilization of Coverage Measurement Tool:** Incorporating tools that measure testing coverage to ensure comprehensive testing of code or requirements is essential.

  Coverage measurement tools are instrumental in ensuring the comprehensive testing of code or requirements. These tools enable the measurement and analysis of test coverage, determining the extent to which the developed code or requirements have been tested (DIAS-NETO et al., 2017; FISCHBACH et al., 2020; WASEEM et al., 2021). The application of code coverage metrics, such as determining the percentage of code tested, allows for control and evaluation of the quality of unit tests (QUESADA-LóPEZ; HERNANDEZ-AGüERO; JENKINS, 2019).

- ❑ **AM-05 Complete Utilization of Continuous Integration Tool:** Integrating constant integration tools to automate test execution and promoting frequent and automated testing within the development process is crucial.

Continuous integration tools for automated tests involve integrating these tools into the development process to automate test execution and promote frequent and automated testing. This integration ensures seamless inclusion of testing activities in the development workflow, with tests executed regularly and automatically (GOTA; GOTA; MICLEA, 2020). Developers can run tests on every code change using continuous integration tools, identifying and addressing issues or bugs early in the development cycle. This approach significantly enhances the software's overall quality and reduces the risk of introducing defects (BRUNETTO et al., 2021a; SPATH, 2023).

- ❑ **AM-06 Suitable Selection of Testing Tools:** Choosing test tools based on the specific features and requirements of the project is a critical aspect of software testing (BRUNETTO et al., 2021a).

Table 3 synthesizes the best practices and their respective categories. In addition, Figure 4 provides a visual representation of the organization of categories and the number of practices defined within each one. This visualization helps to understand the distribution and focus of the practices, offering an intuitive overview of their classification and significance.

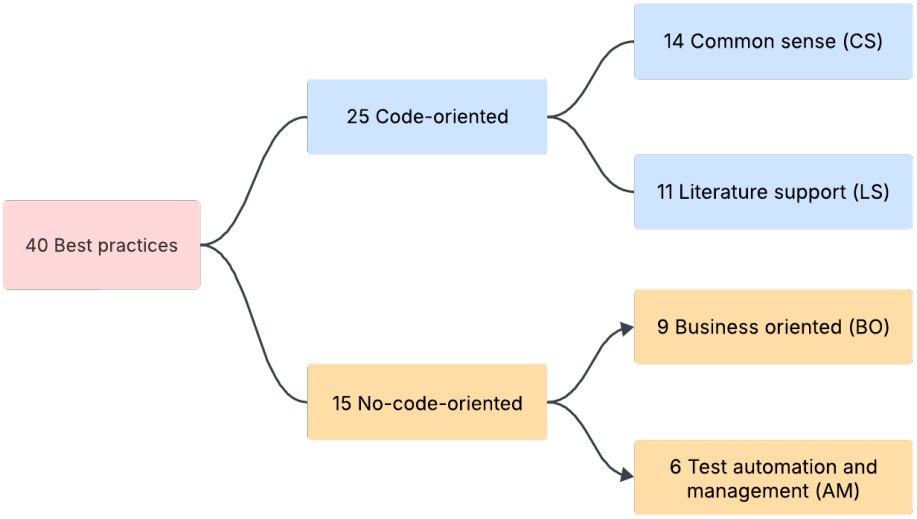

Figure 4 – Good practices classification

3.3 Essential Features of Good Test Cases

This section explores the fundamental features that define high-quality test cases. It builds upon the previously defined practices, emphasizing those that are directly associated with code-oriented activities, such as design, execution, and maintenance of test

Table 3 – Good Practices classified by Category

Group	Category	Good Practices
Code-Oriented	Common Sense (CS)	CS-01, CS-02, CS-03, CS-04, CS-05, CS-06, CS-07, CS-08, CS-09, CS-10, CS-11, CS-12, CS-13, CS-14
	Literature Supported (LS)	LS-01, LS-02, LS-03, LS-04, LS-05, LS-06, LS-07, LS-08, LS-09, LS-10, LS-11
No-Code-Oriented	Business Oriented (BO)	BO-01, BO-02, BO-03, BO-04, BO-05, BO-06, BO-07, BO-08, BO-09
	Test Automation and Management (AM)	AM-01, AM-02, AM-03, AM-04, AM-05, AM-06

cases. While no-code-oriented practices do not directly describe the internal structure of test cases, they provide strategic and managerial support that enhances the overall effectiveness of the testing process. Together, these perspectives offer a comprehensive understanding of what makes a test case not only technically sound, but also sustainable and aligned with project goals.

3.3.1 Ranking of Practices

The ranking of practices provides a structured framework for organizing and prioritizing testing practices based on their relevance and impact. Its primary objective is to offer a clear hierarchy that helps practitioners and researchers identify the most effective practices for achieving high-quality software testing. This framework highlights their specific roles in addressing technical and organizational testing goals by categorizing practices into code-oriented and no-code-oriented.

This ranking is determined by three key criteria: the number of references for each practice, the final score (SC) of the quality assessment of the primary studies supporting them, and the highest H-index of the authors for those studies. These dimensions ensure that the ranking is grounded in empirical evidence and reflects the reliability and relevance of each practice.

Table 4 presents the practices ranking and their respective categories. This analysis highlights the essential features of good test cases, offering valuable insights into their key elements. While the ranking provides a solid foundation for understanding and applying testing practices, it is designed to be flexible and adaptable. Readers are encouraged to adapt it to their specific contexts and needs, whether their focus is on technical precision, resource optimization, or strategic alignment with business goals.

3.3.2 Analysis of Testing Practices

Starting from established best practices that define the essential characteristics of good test cases, an analysis of these practices is conducted to gain valuable insights. This analysis aims to identify practices that naturally complement each other, are related but

not directly dependent, and those with no direct relationship. Through this breakdown, the goal is to improve understanding of how these practices interact and can be applied more effectively in the testing lifecycle in an incremental and complementary way.

Complementary Practices

These practices reinforce or build upon each other naturally, making them integral to the testing lifecycle when used together:

- **CS-02 (Complete Independence of Test Cases)** and **CS-05 (Complete Modularity of Test Cases)**: Independent test cases thrive on modularity. Modularity ensures that test cases are self-contained, reducing potential overlaps or unintended dependencies.
- **CS-01 (Atomic Specification of Test Cases)** and **CS-04 (Boundary Values Analysis)**: Atomic test cases focus on precise units, which pairs well with boundary value analysis, ensuring that edge cases are thoroughly tested.
- **CS-07 (Complex Design for Error Detection)** and **CS-09 (Complete Traceability of Test Cases)**: Complex error detection benefits from traceable test cases, allowing testers to pinpoint failures effectively, essential for debugging intricate systems.
- **CS-13 (Clear Understanding of Test Cases)** and **CS-12 (Regular Review of Test Cases)**: Clear documentation and understanding of test cases ensure that reviews are efficient, allowing reviewers to identify gaps or areas of improvement quickly.
- **LS-03 (Efficient Utilization of Test Code Coverage)** and **LS-05 (Complete Prioritization of Test Cases Design)**: Efficient test coverage ensures that prioritized test cases are covered first, maximizing resource utilization and testing efficiency.
- **LS-01 (Proper Utilization of Test Code Coverage)** and **LS-02 (Required Utilization of Missing Tests)**: Proper coverage highlights missing tests, and identifying those missing tests ensures complete coverage.
- **LS-09 (Deterministic Design of Test Results)** and **LS-10 (Complete Avoidance of Test Side Effects)**: A deterministic test result design ensures consistent outcomes, avoiding side effects that might impact other tests.

Together, these complementarities reinforce the incremental applicability of practices across the testing lifecycle.

Related Practices

These practices overlap or influence each other, though they are not directly dependent:

- **CS-03 (Coverage of Normal and Exceptional Flows)** and **CS-04 (Boundary Values Analysis)**: Boundary value analysis focuses on edge cases, while coverage of regular and exceptional flows ensures that the system works under routine and extreme situations.
- **CS-06 (Detailed Analysis of Size and Complexity)** and **CS-07 (Complex Design for Error Detection)**: Analyzing the size and complexity of the system relates to designing tests capable of detecting errors in complex systems.
- **LS-04 (Small Test Code Generation Footprint)** and **LS-07 (Suitable Utilization of Test Assertions)**: Minimizing the test code footprint and ensuring effective test assertions both contribute to efficient and maintainable testing.
- **LS-06 (Responsible Addition of Test Code Maintenance)** and **CS-08 (Complete Maintenance of Test Code)**: Responsible maintenance ensures that test code stays relevant. However, responsible addition limits what is changed, while complete maintenance ensures tests stay functional.

These relationships highlight overlapping concerns that require coordinated application.

Unrelated Practices

These practices focus on different aspects of testing and don't directly influence each other:

- **CS-09 (Complete Traceability of Test Cases)** and **LS-08 (Responsible Addition of Test Debugging Comments)**: Traceability of test cases ensures accountability while adding debugging comments eases the debugging process but doesn't affect traceability.
- **CS-15 (Structured Coverage of Testing Process)** and **LS-07 (Suitable Utilization of Test Assertions)**: Structured coverage focuses on ensuring all software aspects are tested, while test assertions validate individual conditions. Though both improve quality, they serve different purposes.

Their independence underlines that testing quality depends on addressing distinct dimensions in parallel.

3.4 Tool Support for Test Case Generation

This section explores how various tools are employed to support the generation of test cases, based on the practices identified in the literature. It presents not only the specific tools used across different testing contexts but also the perspectives and insights reported by researchers regarding their benefits, limitations, and impact on the testing process. By analyzing both the technical implementation and practical considerations of these tools, this section highlights their role in enhancing testing efficiency, maintaining software quality, and aligning testing efforts with modern development practices.

3.4.1 Tools Utilized for Test Case Generation

The authors utilize diverse tools to generate adequate test cases for their study. They employ various modeling languages, such as UML and SysML, for system representation (ATIFI; MAMOUNI; MARZAK, 2017). Textual models are also utilized for system description. For automated test execution, frameworks like JUnit and NUnit are used, particularly in agile development contexts (JAHAN et al., 2019). Combinatorial Interaction Testing (CIT) identifies and tests various factor combinations. Tools such as Judy and EclEmma aid in mutation testing and code coverage measurement, respectively (ATIFI; MAMOUNI; MARZAK, 2017). Path Tester generates test cases using the Basis Path Testing method. Alloy Analyzer supports the automatic generation of abstract test cases (GUELFI; RIES, 2008). Tools like Molecule for Ansible and Test Kitchen for Chef are utilized for infrastructure testing (HASAN; BHUIYAN; RAHMAN, 2020). ABT 2.0 is a customized test case generator tailored to the industrial environment (BRUNETTO et al., 2021b). Finally, UMTGtesting is Modederive System test generation, streamlining the process of deriving test cases from system use cases (WANG et al., 2015).

3.4.2 Perspectives on Test Case Generation Tools

Using practical tools in software testing processes is crucial for ensuring the quality and reliability of software products. Various perspectives emerge from the literature within test case generation, shedding light on these tools' utilization, challenges, and benefits.

While test case generation tools are frequently employed in testing processes, a consensus exists regarding their underutilization in critical supporting activities such as test planning and defect prevention. This suggests significant potential for improvement in these areas. Agile testing methodologies are widely recognized as advantageous for software development, paving the way for further exploration and advancement of automation software testing tools.

Test automation tools, notably Selenium, are highly regarded for their invaluable contributions to testing activities, reflecting a generally positive sentiment toward test

automation. Automation is deemed indispensable for enhancing reliability and cost-effectiveness in software development. However, it is emphasized that the effectiveness of these tools hinges on the presence of an optimized development process.

Specific tools like MISTA and UMTG stand out for their efficacy in simplifying the testing process and automating test case generation, offering promising solutions to streamline testing efforts. Despite facing challenges such as language dependency and the absence of robust oracles, automated test case generation is acknowledged as a viable approach to reduce testing effort, improve efficiency, and improve testing quality, particularly in safety-critical systems.

3.5 Final Considerations

This chapter provided a comprehensive identification, refinement, and classification of good software testing practices based on an extensive systematic review of the literature. Starting with 131 practices initially extracted, a rigorous analytical process led to the consolidation of 40 refined practices categorized into code-oriented and no-code-oriented groups. These practices were also semantically and structurally analyzed to ensure their applicability, relevance, and clarity to guide the development of high-quality test cases. Beyond the refinement itself, this chapter also provides a structured framework for the classification and prioritization of practices, which serves as a conceptual foundation for organising knowledge on test case design in a systematic and replicable manner.

The categorization into common sense and literature-supported practices (for code-oriented) and business-oriented and test automation/management practices (for no-code-oriented) offers a multifaceted view of software testing strategies. Furthermore, the ranking and relational analysis of practices highlight their practical utility, interdependence, and areas of synergy, helping practitioners implement them effectively and incrementally in real-world scenarios.

By synthesizing this refined knowledge base, the chapter not only addresses the gap in test case design practices but also lays the foundation for intelligent support systems in software testing. These insights serve as the theoretical and methodological groundwork for the development of TAI-EvalGenTCS, an AI-powered tool that operationalizes these practices by evaluating and improving test case quality.

The definition and classification of these practices directly support the research hypothesis that a structured set of best practices can guide automated tools to enhance test case design quality. In addition, the empirical validation of these practices with industry professionals, presented in Chapter 5, further demonstrates their applicability and strengthens their relevance in the real world.

The next chapter introduces TAI-EvalGenTCS, a tool explicitly designed to operationalize these practices. By embedding this consolidated knowledge into its architecture,

the tool bridges the gap between validated best practices and their automated application. It illustrates how artificial intelligence can bridge the gap between test automation and test quality by ensuring a consistent application of good testing practices.

Table 4 – Practice classification

Practice	Nº of Ref.	SC1	Year	HI1	SC2	Year	HI2	SC3	Year	HI3	SC4	Year	HI4	SC5	Year	HI5	SC6	Year	HI6	SC7
CS-02 Complete Independence of Test Cases	7	9	2006	6	6	2017	23	9	2020	33	6	2021	49	9	2021	19	9	2022	15	3
CS-13 Clear Understanding of Test Cases	7	9	2006	6	9	2017	47	6	2018	24	6	2019	23	9	2022	15	-	-	-	-
CS-15 Structured Coverage of Testing Process	6	9	2006	6	6	2018	20	9	2020	33	6	2022	12	9	2023	-	6	2023	52	-
CS-06 Detailed Analysis of Size and Complexity	5	9	2006	6	9	2012	12	9	2020	33	6	2021	49	9	2022	15	-	-	-	-
CS-01 Atomic Specification of Test Cases	4	9	2006	6	6	2012	69	6	2013	34	-	-	-	-	-	-	-	-	-	-
CS-03 Coverage of Normal and Exceptional Flows	4	9	2006	6	6	2019	1	6	2021	49	9	2022	15	-	-	-	-	-	-	-
CS-04 Boundary Values Analysis	4	6	2021	49	9	2022	15	9	2022	50	6	2023	-	-	-	-	-	-	-	-
LS-03 Efficient Utilization of Test Code Coverage	4	9	2017	47	9	2017	44	9	2019	62	6	2021	36	-	-	-	-	-	-	-
LS-05 Complete Prioritization of Test Cases Design	4	9	2006	6	9	2017	47	6	2022	-	6	2022	7	-	-	-	-	-	-	-
LS-08 Responsible Addition of Test Debugging Comments	4	9	2001	27	9	2017	47	9	2019	62	9	2020	33	-	-	-	-	-	-	-
CS-05 Complete Modularity of Test Cases	3	9	2006	6	11	2019	22	6	2021	49	-	-	-	-	-	-	-	-	-	-
CS-10 Strict Use of Performance and Security Testing	3	7	2018	47	9	2019	13	9	2022	15	-	-	-	-	-	-	-	-	-	-
CS-12 Regular Review of Test Cases	3	9	2006	6	9	2017	47	9	2017	44	-	-	-	-	-	-	-	-	-	-
CS-16 Complete Assurance of Test Code Quality	3	11	2019	98	9	2020	33	9	2022	15	-	-	-	-	-	-	-	-	-	-
LS-06 Responsible Addition of Test Code Maintenance	3	9	2006	6	9	2017	47	6	2021	49	-	-	-	-	-	-	-	-	-	-
LS-09 Deterministic Design of Test Results	3	9	2006	6	9	2017	47	9	2019	62	-	-	-	-	-	-	-	-	-	-
CS-07 Complex Design for Error Detection	2	11	2019	22	9	2020	33	-	-	-	-	-	-	-	-	-	-	-	-	-
CS-08 Complete Maintenance of Test Code	2	9	2019	13	9	2022	15	-	-	-	-	-	-	-	-	-	-	-	-	-
CS-09 Complete Traceability of Test Cases	2	11	2019	98	9	2021	19	-	-	-	-	-	-	-	-	-	-	-	-	-
LS-01 Proper Utilization of Test Code Coverage	2	9	2017	47	9	2017	20	-	-	-	-	-	-	-	-	-	-	-	-	-
LS-02 Required Utilization of Missing Tests	2	9	2017	47	7	2020	30	-	-	-	-	-	-	-	-	-	-	-	-	-
LS-04 Small Test Code Generation Footprint	2	9	2006	6	9	2017	47	-	-	-	-	-	-	-	-	-	-	-	-	-
LS-07 Suitable Utilization of Test Assertions	2	9	2017	47	6	2021	49	-	-	-	-	-	-	-	-	-	-	-	-	-
LS-10 Complete Avoidance of Test Side Effects	2	9	2006	6	9	2017	47	-	-	-	-	-	-	-	-	-	-	-	-	-
LS-11 Suitable Utilization of Test Labels and Categories	2	9	2006	6	9	2017	47	-	-	-	-	-	-	-	-	-	-	-	-	-

Chapter 4

TAI-EvalGenTCS: An AI-Based Tool for Evaluation and Generation of Test Case Sets Based on Good Software Testing Practices

This chapter presents TAI-EvalGenTCS, a tool designed to evaluate and generate test cases based on good software testing practices. The tool integrates OpenAI's GPT-4-turbo model to assess test case quality, suggest improvements, and provide AI-generated test cases. It aims to bridge the gap between automation and test quality by focusing not only on execution, but also on test design principles such as maintainability, modularity, and completeness.

4.1 Motivation and Context

4.1.1 The Need for High-Quality Test Cases

Ensuring high-quality software testing practices is an essential aspect of modern software development (MULLA, 2024; KHATAMI; BRANDT; ZAIDMAN, 2024). Poor test case design can lead to undetected defects, increased maintenance costs, and unreliable software systems (CATAL, 2012; RAO et al., 2022a; BARRAOOD; MOHD; BAHAROM, 2022). Although various tools automate test execution and management, specific solutions to analyze the quality of test cases themselves are lacking (Katalon, Inc., 2025; PractiTest Ltd., 2025; SmartBear Software, 2025; Ranorex GmbH, 2025; Gurock Software, 2025).

The quality of software testing directly influences the reliability and maintainability of software systems (JIA, 2023). Despite advances in test automation, the quality of

individual test cases is often overlooked (BAQAR; KHANDA, 2024; ALEKSANDROVA; DOBRYNINA, 2024). Poorly designed test cases lead to ineffective testing, increased debugging efforts, and ultimately software defects that persist into production (PONTILLO, 2024; DU, 2023). The motivation behind the development of TAI-EvalGenTCS stems from the need to bridge this gap by providing an AI-driven approach to the evaluation and optimization of test cases.

Test cases serve as the backbone of software verification processes, ensuring that applications function as expected in different scenarios (NAIR, 2023). However, writing effective test cases requires adherence to best practices that guarantee clarity, modularity, maintainability, and coverage. In many organizations, these best practices are not systematically enforced, leading to:

- ❑ **Redundant or Inefficient Test Cases**: Many test cases test overlapping functionality, introducing unnecessary maintenance overhead.
- ❑ **Unmaintainable Test Code**: Poorly structured test cases make it difficult to adapt them to evolving software requirements.
- ❑ **Limited Coverage and Effectiveness**: Without structured best practices, test suites may not cover edge cases, leading to undiscovered defects.

The motivation for TAI-EvalGenTCS was to address these challenges by providing an automated solution that evaluates test cases based on a structured set of best practices and offers AI-driven improvements.

4.1.2 Bridging the Gap Between Automation and Quality

Existing test automation tools focus on executing test cases efficiently but do not assess the quality of the test cases themselves. Automated test execution ensures that predefined test cases run consistently across environments, but it does not validate whether the test cases follow best practices in terms of readability, maintainability, and efficiency.

TAI-EvalGenTCS can fill this gap by:

- ❑ **Assessing Compliance**: Evaluating test cases based on established best practices.
- ❑ **Providing Actionable Insights**: Feedback on areas where test cases can be improved.
- ❑ **Generating Optimized Test Cases**: Using AI to rewrite test cases while maintaining their intended functionality.

This functionality enhances software testing workflows by ensuring that test cases are not only executed correctly, but also contribute to long-term software maintainability.

4.1.3 Leveraging AI for Intelligent Test Case Evaluation

The adoption of AI in software engineering has opened new possibilities to automate complex processes that traditionally required human expertise. TAI-EvalGenTCS leverages OpenAI's GPT-4-turbo model to:

- ❏ Analyze test cases across different programming languages.
- ❏ Evaluate compliance with structured best practices.
- ❏ Suggest modifications to improve the clarity and maintainability of the test case.

By applying AI-driven insights, the tool enables development teams to standardize test case quality, reduce human effort in review processes, and enforce testing standards across projects.

4.2 TAI-EvalGenTCS Tool

TAI-EvalGenTCS is an AI-powered tool designed to evaluate and enhance the quality of test cases based on well-established best practices. By leveraging artificial intelligence, it provides an automated and objective assessment of test case implementations, identifying strengths and areas for improvement according to structured evaluation criteria. The tool not only offers a compliance score but also generates optimized test cases that adhere to high-quality standards, thereby improving software testing effectiveness and maintainability.

4.2.1 Architecture

TAI-EvalGenTCS consists of a modular architecture built using modern web and AI technologies. The system is composed of the following key components:

- ❏ **Backend:** Developed in Node.js (version 18) with Express, it serves as the core processing unit, handling API requests, evaluating test cases, and interfacing with the AI model.
- ❏ **Frontend:** Implemented in Vue 3, it provides an intuitive interface for users to input test case code, trigger evaluations, and review detailed feedback.
- ❏ **AI Model:** Utilizes OpenAI's GPT-4-turbo, instantiated and parameterized with the best practices outlined in Chapter 3. Rather than being retrained, the model is prompted and configured to evaluate and optimize test cases based on these practices using OpenAI's assistant technologies.

The architecture follows a client-server model, where the front end allows user interaction while the back end processes test case evaluations and manages AI-driven responses.

The full implementation of TAI-EvalGenTCS is publicly available and divided into two repositories:

- Frontend (Vue 3): <https://github.com/caviva/test-case-evaluator.git>
- Backend (Node.js/Express): <https://github.com/caviva/chat-practices.git>

4.2.2 Main Functionalities

TAI-EvalGenTCS provides a seamless experience for test case evaluation and improvement through the following functionalities:

- **Test Case Evaluation:** Users can paste a test case into an input field and click the "Evaluate" button to initiate the analysis.
- **Best Practice Compliance Scoring:** The tool provides a compliance score (in percentage) indicating how well the test case adheres to best practices.
- **Detailed Practice Analysis:** Each best practice is evaluated individually, categorized as Compliant (green), Non-compliant (red), or Not Applicable (gray). A corresponding explanation is provided for each result, allowing users to understand the strengths and weaknesses of their test case.
- **AI-Generated Optimized Test Case:** The AI generates an improved version of the test case, making its best effort to adhere to the best practices while maintaining the original function of the code provided by the user. This ensures that the optimized test case remains relevant and useful in the context of the software being tested.

The interface is designed to be minimalistic, yet powerful. It allows for quick feedback on test case quality and practical recommendations for improvement.

4.2.3 Validation of the Tool

To assess the effectiveness and reliability of TAI-EvalGenTCS, a validation study was carried out using a set of 16 Java projects. These projects were selected to represent diverse implementations of data structures and algorithms, ensuring a broad evaluation across different codebases.

4.2.3.1 Experimental Setup

Each of the 16 Java projects was subjected to an automated test case generation process using EvoSuite, a widely recognized tool to generate unit tests based on evolutionary algorithms (FRASER; ARCURI, 2011). The test cases produced by EvoSuite were then evaluated using TAI-EvalGenTCS, which assigned a compliance score based on the adherence to best practices. The original test cases served as a baseline measurement of quality.

4.2.3.2 Results and Analysis

The evaluation results, summarized in Table 5, indicate that, in the majority of cases, the AI-generated test cases exhibited significant improvements in quality. The compliance scores increased in 12 out of the 16 projects, demonstrating that TAI-EvalGenTCS effectively enhances test cases according to best practices. Specific improvements observed included:

- ❏ Reduction in redundant assertions, making test cases more concise and focused.
- ❏ Enhanced modularity, reduced dependency between test cases, and facilitated independent execution.
- ❏ Improved boundary value analysis, ensuring that edge cases were tested more thoroughly.

Although most of the test cases showed positive improvements, four projects exhibited marginal score variations. This was attributed to the complexity of their initial test cases, which already adhered to high-quality standards. However, even in these cases, the AI-generated test cases maintained or slightly refined structural aspects without degrading quality.

These findings validate TAI-EvalGenTCS as a reliable tool for enhancing test case quality. The AI-generated refinements consistently aligned with best practices, making test cases more maintainable and effective in detecting potential defects.

4.3 Example Usage and Demonstration

A user-friendly web interface enables users to evaluate test cases efficiently. The evaluation process involves the following.

1. Pasting a test case code snippet into the provided input field. The interface allows users to input their test case code easily, as illustrated in Figure 5.

Project	Code Score	Tool Score	Difference
01Max	48%	56%	+8%
02MaxMin A	50%	56%	+6%
03MaxMin B	50%	58%	+8%
04MaxMin C	43%	64%	+21%
05Sorting	36%	60%	+24%
06Fibonacci A	44%	60%	+16%
07Fibonacci B	44%	48%	+4%
08MaxMin	56%	57%	+1%
09Sorting	48%	52%	+4%
10Evaluate Matrix	44%	36%	-8%
11List Array	52%	55%	+3%
12List	48%	50%	+2%
13Stack A	52%	64%	+12%
14Stack B	56%	52%	-4%
15Queue A	56%	52%	-4%
16Queue B	56%	52%	-4%

Table 5 – Comparison of Test Code Scores and AI-EvalGenTCS Scores with Differences

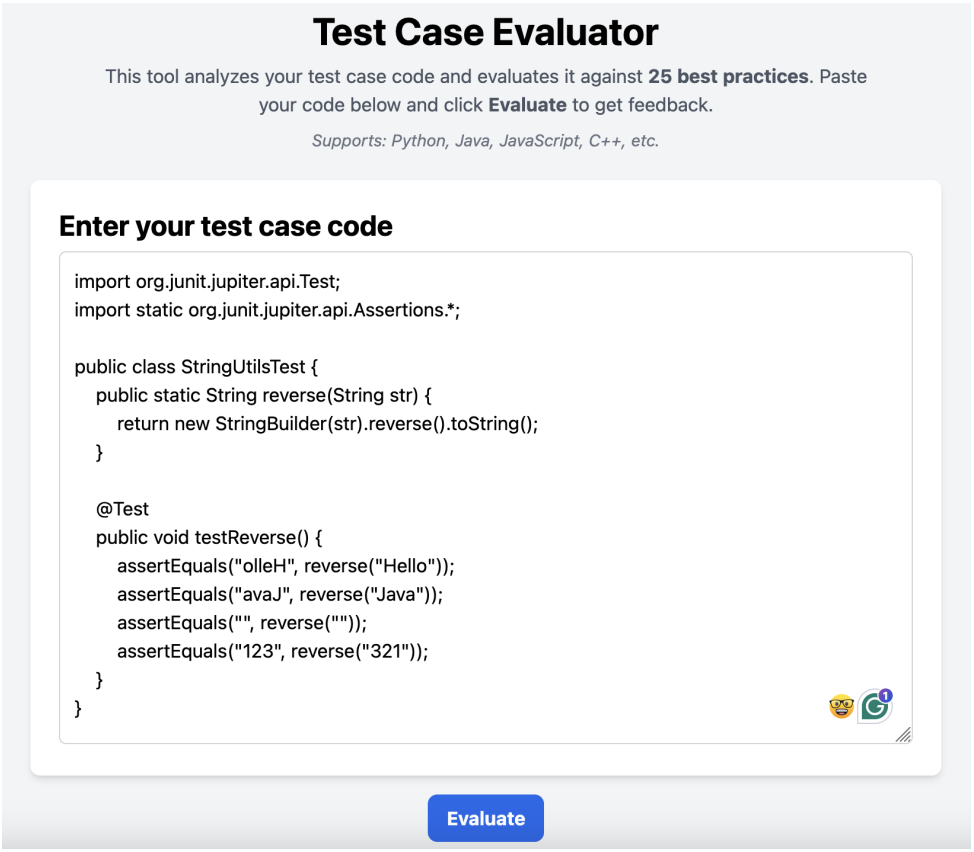

Figure 5 – Test case input field

2. Clicking the “Evaluate” button, which sends the code to the backend for analysis. Once the button is pressed, the evaluation process is triggered, as shown in Figure 6.

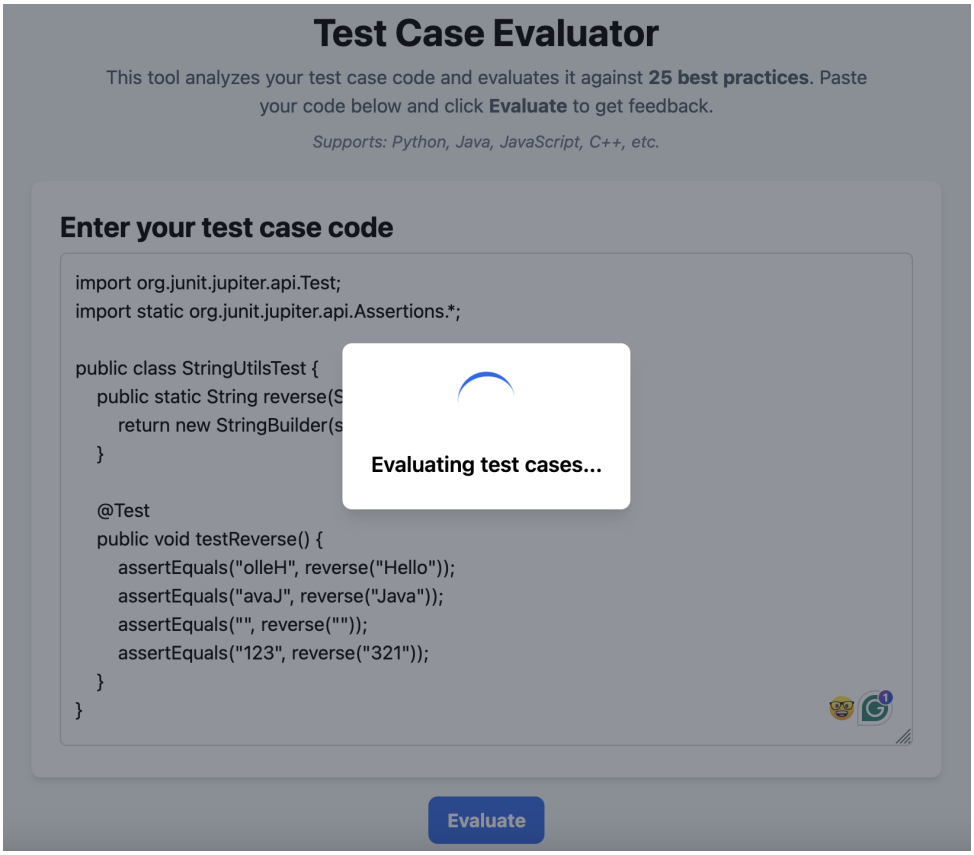

Figure 6 – Evaluation process initiation

3. The AI processes the request and returns a response containing:
   - ❏ Compliance score in percentage.
   - ❏ A breakdown of each best practice and its status.
   - ❏ A color-coded summary highlighting compliance results.
   - ❏ An AI-generated optimized test case that improves upon the original code while maintaining its intended functionality.

   The evaluation results are displayed in a structured format across three sections:

4. Users can review and copy the improved test case code for future use. The generated optimized test case is available for copying, ensuring that users can directly apply the recommended improvements, as seen in Figure 10.

4.3.1 Type of License

TAI-EvalGenTCS follows an open-access licensing model to promote transparency, collaboration, and extensibility. The tool is released under the MIT License, which gives users the freedom to use, modify, and distribute the software with minimal restrictions.

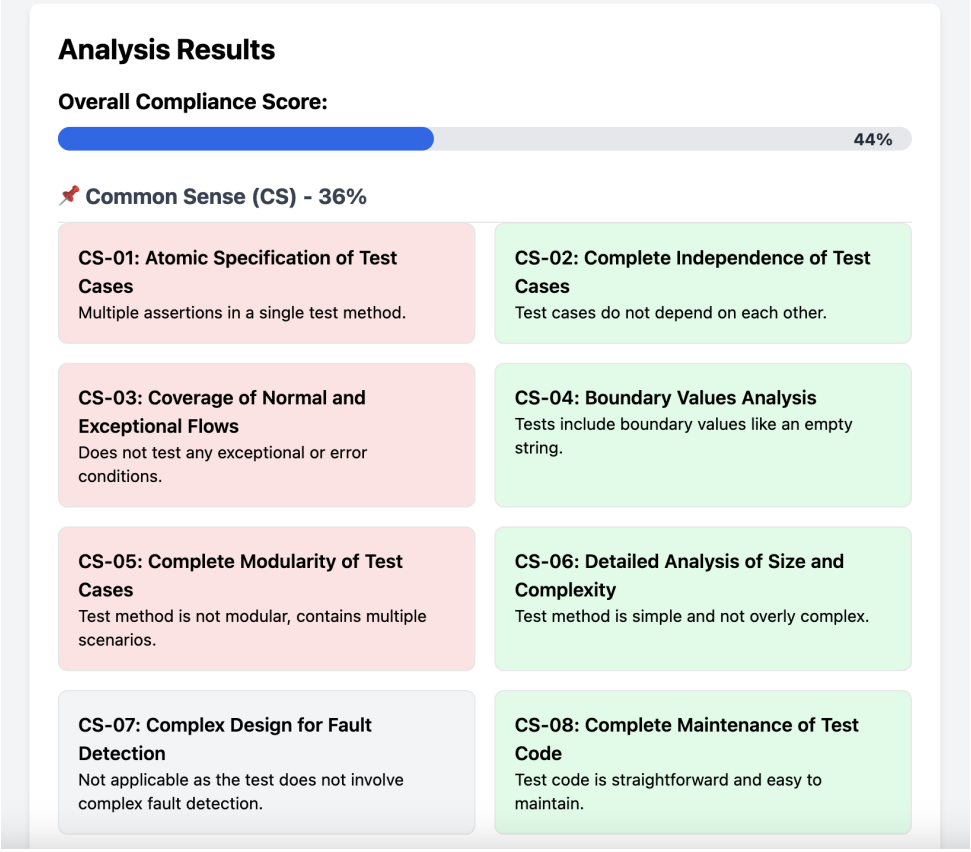

Figure 7 – Evaluation results - Part 1: Compliance Score

4.3.1.1 Key Aspects of the MIT License

The MIT license is widely used in the software industry due to its permissive nature (SALTZER; WALDEN, 2020). The key aspects of this license include:

- **Free Usage**: Users are allowed to use the software for personal, academic, or commercial purposes without restrictions.
- **Modification and Distribution**: Developers can modify the source code and distribute it as part of their projects, open-source or proprietary.
- **Attribution Requirement**: Any redistribution of the tool must include the original copyright notice and the license terms.
- **Liability and Warranty Disclaimer**: The software is provided "as is," without warranties or guarantees of performance or suitability for any particular purpose.

4.3.1.2 Implications for the Software Testing Community

By adopting the MIT license, TAI-EvalGenTCS aims to:

- Encourage researchers and developers to enhance the tool by integrating new best practices or refining evaluation models.

Figure 8 – Evaluation results - Part 2: Best Practice Breakdown

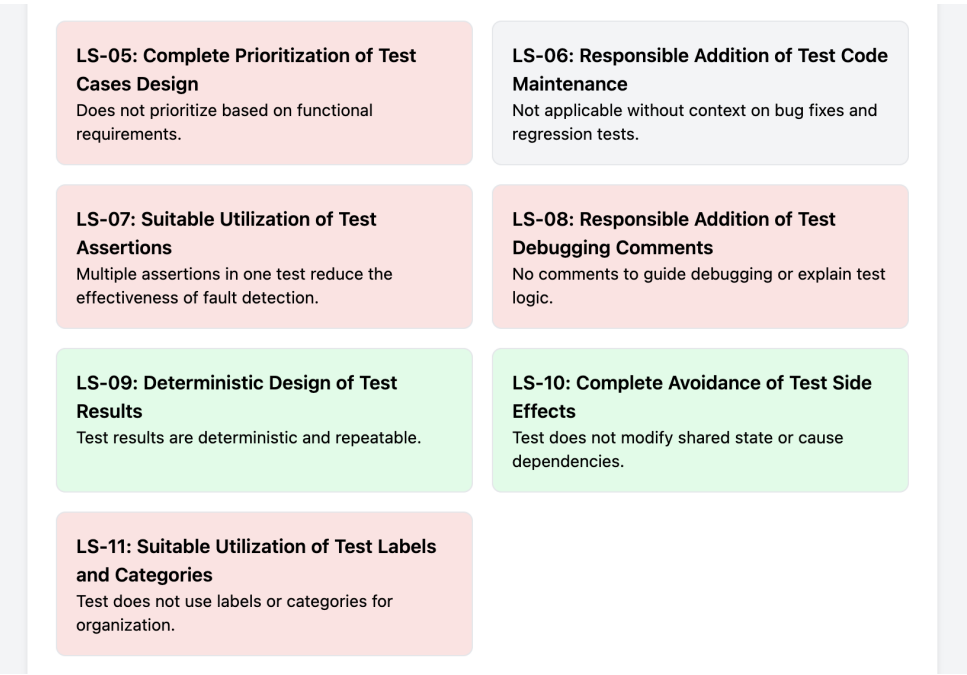

Figure 9 – Evaluation results - Part 3: Best Practice Breakdow

❑ Facilitate industry adoption by providing a simple and business-friendly license approach.

Figure 10 – AI-generated optimized test case

- ❑ Support academic and educational initiatives where software testing tools are essential for teaching and research.

This licensing approach ensures that TAI-EvalGenTCS remains accessible while fostering innovation and collaborative improvements in software testing methodologies.

4.4 Comparative Analysis of Related Tools

In the evolving landscape of software testing, various tools have been developed to enhance the efficiency and effectiveness of testing processes. This section provides a detailed comparative analysis of prominent test automation and management tools, highlighting their features, advantages, and disadvantages, and comparing them with TAI-EvalGenTCS.

4.4.1 Katalon Studio

Overview: Katalon Studio is a comprehensive test automation platform that supports web, mobile, desktop, and API testing (Katalon, Inc., 2025; EREIZ, 2019). It offers a flexible environment suitable for both beginners and experienced testers, providing no-code, low-code, and scripting modes.

Advantages:

- ❏ Supports multiple types of applications, including web, mobile, desktop, and APIs.
- ❏ Offers no-code and low-code options, making it accessible to users with varying coding skills.
- ❏ Seamlessly integrates with CI/CD pipelines and various tools, enhancing workflow automation.

Disadvantages:

- ❏ May require significant system resources, potentially affecting performance on lower-end machines.
- ❏ Advanced features may present a learning curve for users without prior automation experience.

Comparison with TAI-EvalGenTCS: While Katalon Studio focuses on automating and executing tests across various platforms, TAI-EvalGenTCS specializes in evaluating the quality of test cases based on best practices. Katalon provides a broader automation framework, whereas TAI-EvalGenTCS offers in-depth analysis and optimization of test case code.

4.4.2 PractiTest

Overview: PractiTest is a test management platform designed to provide complete visibility and control over the testing process (PractiTest Ltd., 2025). It supports manual and automated testing, offering integration with various bug tracking and automation tools.

Advantages:

- ❏ Offers end-to-end test management capabilities, including requirements, tests, and issue tracking.
- ❏ Highly customizable to fit different workflows and methodologies.
- ❏ Integrates with numerous tools such as JIRA, Selenium, and Jenkins, facilitating a seamless workflow.

Disadvantages:

- ❏ Pricing may be a consideration for smaller teams or organizations with limited budgets.
- ❏ The extensive features might be overwhelming for teams with simple testing needs.

Comparison with TAI-EvalGenTCS: PractiTest focuses on managing the testing lifecycle, from planning to execution and tracking. In contrast, TAI-EvalGenTCS is dedicated to assessing and improving the quality of test case code, offering AI-generated optimizations.

4.4.3 TestComplete

Overview: TestComplete by SmartBear is an automated UI testing tool for web, desktop, and mobile applications (SmartBear Software, 2025). It supports various scripting languages and integrates with CI/CD tools to facilitate continuous testing.

Advantages:

- ❏ Capable of testing a wide range of applications across different platforms.
- ❏ Supports multiple scripting languages, including JavaScript, Python, and VBScript.
- ❏ Utilizes AI-powered object recognition to improve test stability and maintenance.

Disadvantages:

- ❏ Licensing can be expensive, which may not be suitable for smaller organizations.
- ❏ Advanced features may require significant expertise to utilize effectively.

Comparison with TAI-EvalGenTCS: TestComplete is geared towards automating the execution of tests across various platforms, whereas TAI-EvalGenTCS focuses on evaluating and enhancing the quality of test case code through AI-driven analysis.

4.4.4 Ranorex Studio

Overview: Ranorex Studio is an all-in-one test automation tool for desktop, web, and mobile applications (Ranorex GmbH, 2025). It offers a user-friendly interface with codeless click-and-go automation, as well as a full IDE for advanced users.

Advantages:

- ❏ Provides codeless automation, making it accessible to testers without programming skills.
- ❏ Includes features like robust reporting, data-driven testing, and cross-browser testing.
- ❏ Integrates with CI/CD tools and supports various technologies and platforms.

Disadvantages:

- ❏ May experience slower performance with large test suites.

❏ The pricing structure may be a barrier for small teams or startups.

Comparison with TAI-EvalGenTCS: Ranorex Studio emphasizes automating test execution across multiple platforms, while TAI-EvalGenTCS specializes in analyzing and improving test case code quality through adherence to best practices.

4.4.5 TestRail

Overview: TestRail is a test case management and orchestration platform that centralizes testing efforts, integrates with various tools, and provides analytics to optimize testing processes (Ranorex GmbH, 2025; FEDOSOV et al., 2020).

Advantages:

❏ Provides a unified platform for managing test cases, plans, and runs.

❏ Seamlessly integrates with issue trackers, automation tools, and CI/CD pipelines.

❏ Offers detailed reporting and analytics to monitor testing progress and quality.

Disadvantages:

❏ New users may require time to adapt to the platform's features and workflows.

❏ Subscription-based pricing may be a consideration for budget-conscious teams.

Comparison with TAI-EvalGenTCS: TestRail focuses on managing and organizing test cases and testing activities, while TAI-EvalGenTCS is dedicated to evaluating the quality of the test case code and providing AI-driven optimizations.

4.5 Final Considerations

TAI-EvalGenTCS is an innovative tool that enables the evaluation and improvement of test cases based on good software testing practices. Artificial intelligence delivers value beyond traditional automation. Its modular architecture, user-friendly interface, and ability to generate optimized test cases position it as a concrete contribution to software quality enhancement.

The development of TAI-EvalGenTCS is a direct step towards testing the research hypothesis by translating the defined best practices into an AI-driven evaluation and generation process.

The validation results using 16 Java projects confirmed that the practice-based approach leads to significant improvements in clarity, modularity, and coverage. Compared to other tools, TAI-EvalGenTCS stands out as a specialized solution for qualitative analysis of test case design.

To ensure that the set of good testing practices used by TAI-EvalGenTCS truly reflects the needs of the software industry, the next chapter presents an empirical validation conducted through a structured survey with experienced practitioners. This upcoming study is essential to confirm the relevance, clarity and applicability of the proposed practices. It reinforces the scientific foundation of the tool and strengthens its alignment with real-world software development environments, ultimately bridging theory and professional practice.

Chapter 5

Validating Good Software Testing Practices: A Practitioner-Centered Survey

This chapter presents the study related to the design and execution of a survey aimed at validating the set of good software testing practices identified in the previous chapter. The survey was conducted with experienced practitioners to collect their perceptions regarding the relevance, clarity, and applicability of each proposed practice. The findings contributed to refining the set of practices and confirming their alignment with the practical needs and realities of the software industry.

5.1 Introduction

The definition of good software testing practices constitutes a critical foundation for ensuring the effectiveness and maintainability of test cases. While Chapter 3 introduced a consolidated set of 40 practices based on a rigorous literature review, and Chapter 4 demonstrated the application of those practices through the TAI-EvalGenTCS tool, it remains essential to validate their practical relevance with experienced professionals. This chapter addresses this need by presenting the design, implementation, and results of a practitioner-centered survey.

The objective of this survey was to evaluate the relevance, clarity, and applicability of the 40 proposed practices, with a specific focus on the 25 code-oriented practices that directly influence the technical creation and execution of test cases. These practices were initially extracted from 103 primary studies and refined into a structured classification that includes Common Sense and Literature-Supported categories.

A multistage process was followed to ensure the quality and validity of the survey instrument. First, the survey was reviewed by a group of academic researchers with expertise in software testing and empirical studies. Their feedback was used to refine the questions, descriptions, and user interface. Subsequently, a pilot version of the survey was conducted with a small group of academic and industry practitioners to identify usability issues and ensure content clarity. Finally, the validated version of the survey was applied to a sample of 25 experienced software professionals from various countries and experience levels.

The survey was conducted through a custom-developed dedicated web application. This custom tool was designed to support the specific structure and logic required by the evaluation process, including conditional visualizations, stepwise progression, and support for structured qualitative feedback. The tailored nature of the tool ensured a smooth experience for the respondents, allowing for the collection of detailed and relevant data for analysis.

5.2 Survey Design and Implementation

The survey was designed to validate a curated set of 40 good software testing practices through a structured evaluation by practitioners. The emphasis was placed on the 25 code-oriented practices, which are directly related to the creation, execution, and quality assurance of test cases. The practices were previously classified into two main categories: Common Sense Practices and Literature-Supported Practices, both of which form the basis for the quality assessment of code-level tests. The complete set of questions in the survey is available at the Appendix A.

To address different dimensions of practitioner insight, the survey was structured into four consecutive steps. Each step was designed to collect complementary perspectives on the proposed practices and to collect quantitative and qualitative data.

- **Step 1 – Practice Evaluation:** Participants were asked to evaluate 25 code-oriented practices using a three-point Likert scale: Agree, Neutral, or Disagree. An open text field was also provided for optional qualitative observations.
- **Step 2 – Practice Relevance to Category:** Participants were asked to assess how well each of the 25 code-oriented practices aligns with its assigned classification — either Common Sense or Literature-Supported. This step aimed to validate whether the theoretical categorization reflected the perceptions of the practitioners. Participants could also provide qualitative feedback to justify or question categorization.

- ❏ **Step 3 – Practice Prioritization:** Participants were asked to rank the 25 code-oriented practices in importance order according to their experience and perceived impact in software testing contexts. The practices were presented in random order to prevent positional bias.
- ❏ **Step 4 – Tool Support Mapping:** A list of 27 code-oriented practices (including extended ones) was presented. Participants indicated whether they were aware of any tools that support each practice. If so, they were invited to name the tool or provide a relevant URL.

The system was designed with usability in mind, ensuring that participants could complete each step without friction. Progress indicators and response confirmation mechanisms were included to improve the user experience. Crucially, the application enforced a linear progression model, preventing participants from revisiting previous steps once they had been submitted, thereby ensuring the integrity of the responses.

In general, this approach enabled the systematic collection of structured high-quality data to analyze the practical perception and prioritization of test case design practices.

The complete implementation of the survey system, including both backend and frontend components, is openly available to ensure transparency and reproducibility. The source code can be accessed at the following repositories:

- ❏ Backend: <https://github.com/caviva/surveyBP.git>
- ❏ Frontend: <https://github.com/caviva/survey-ufscar.git>

These repositories contain the full codebase and instructions necessary to deploy and replicate the survey, as well as to extend its functionalities for future empirical studies.

5.2.1 Type of License

The survey system follows an open-access licensing model to foster transparency, reproducibility, and further research. The source code is released under the MIT License, which allows users to freely use, modify, and distribute the software with minimal restrictions. This licensing choice reinforces the intention of supporting collaboration between academia and industry.

5.3 Participant Profile

The survey had 25 participants. The participants represented a diverse cross section of the software industry and academia, bringing varied perspectives on software testing practices. The selection process was designed to ensure relevance, diversity, and practical experience among the respondents.

Before the public distribution of the survey, two critical validation stages were performed to refine its structure and ensure clarity.

- **Expert Review:** The initial survey draft was reviewed by a panel of academic researchers and faculty members with expertise in software testing. Feedback was incorporated to improve the formulation of the questions, eliminate ambiguity, and align the evaluation dimensions with current research.
- **Pilot Testing:** A pilot version was administered to a smaller group of academic and industry practitioners. This helped identify usability issues and refine the flow, layout, and terminology used in the survey interface.

After these adjustments, the final version of the survey was distributed to experienced professionals with a proven background in software testing. Participants were required to complete all four steps in a single session, ensuring data consistency and completeness.

The demographic and professional profile of the participants is summarized below.

- **Geographic Distribution:** Participants were located in multiple countries, contributing international perspectives to the evaluation of practices.
- **Professional Roles:** Respondents included QA Engineers, Test Automation Engineers, Software Developers in Test (SDETs), QA Managers, and Software Architects, ensuring a broad understanding of testing activities.
- **Academic Background:** The majority of participants held at least a bachelor's degree, with several possessing Master's or Doctoral-level education in fields related to Computer Science or Software Engineering.
- **Experience Level:** Participants had an average of more than 5 years of experience in software testing, and several reported 10+ years in the field.
- **Gender and Age Distribution:** The sample included both male and female participants in a diverse age range, from early career professionals to senior experts.

This rich and experienced pool of respondents adds validity and depth to the survey findings. The diversity of roles, backgrounds, and perspectives contributes to a well-rounded understanding of how good software testing practices are perceived, prioritized, and supported in real-world testing environments.

5.4 Survey Results

This section presents a detailed analysis of the results obtained from the practitioner-centered survey. The survey was structured into four main steps, each designed to gather

specific types of feedback on the 25 code-oriented testing practices introduced in the previous chapter. The results reported here aim to provide empirical insight into how experienced professionals perceive, evaluate, prioritize, and associate tool support with these practices.

The analysis includes both quantitative and qualitative data. It highlights overall agreement levels, relevance scores, prioritization trends, and tool usage patterns, providing a comprehensive view of the current state of practice in software testing as perceived by industry experts.

5.4.1 Practice Evaluation

The first step of the survey was to capture the initial reactions of the practitioners to each of the 25 code-oriented practices. Participants were asked to rate each practice using a 3-point Likert scale: *Agree*, *Neutral*, or *Disagree*. In addition, optional open-text fields allowed respondents to provide qualitative observations or justifications for their ratings.

The aggregated results reveal a strong level of agreement in most practices. Practices such as **CS-01 (Atomic Specification of Test Cases)**, **CS-04 (Boundary Values Analysis)**, and **CS-13 (Clear Understanding of Test Cases)** received near-universal agreement, confirming their foundational role in test design. In contrast, practices like **LS-02 (Required Utilization of Missing Tests)** and **LS-04 (Small Test Code Generation Footprint)** exhibited a higher proportion of neutral or disagreed responses, suggesting either lack of clarity or limited applicability in certain contexts.

Qualitative comments enriched the analysis and provided valuable insight for the practitioner. For example, a respondent stated:

> "Atomic test cases ensure the smallest components on the system work as expected. When each small part works well, it gives confidence that the whole system will also function correctly."

Another participant highlighted concerns over excessive modularization.

> "The level of modularity might need to be adapted based on the complexity and scope of the system under test."

These comments emphasize that while best practices are generally accepted, their application must be contextualized.

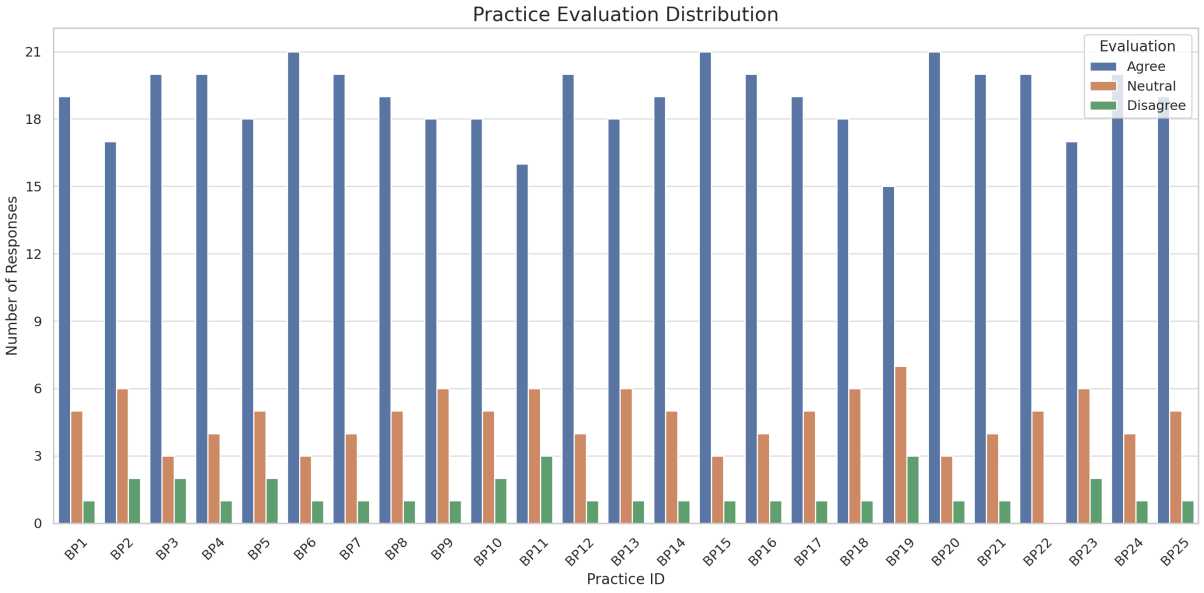

Figure 11 – Distribution of responses for each code-oriented practice

As shown in Figure 11, the majority of the proposed practices received a strong consensus of agreement among participants. Practices such as BP-01 Atomic Specification, BP-02 Complete Independence, and BP-05 Test Code Quality received nearly unanimous support. On the other hand, practices like BP-03 Coverage of Exceptional Flows and BP-24 Avoidance of Side Effects show slightly higher proportions of neutral or disagree responses, suggesting either lower consensus or potential need for clarification. This distribution helps to visually identify which practices are universally accepted and which may benefit from further refinement or contextual adaptation.

Overall, this first step confirmed that the selected practices are generally aligned with practitioner expectations, especially those grounded in long-established testing principles. The qualitative feedback also provided opportunities to clarify the definitions and intended scope of more ambiguous practices.

5.5 Practice Relevance Analysis

In this step of the survey, participants were asked to assess the relevance of each of the 25 code-oriented practices with respect to their assigned classification: either as a Common Sense (CS) practice or as a Literature-Supported (LS) practice. The objective was to verify whether practitioners found the classification coherent based on their experience and intuition.

The evaluation used a four-point scale: *Very High*, *High*, *Moderate*, and *Low*. The analysis focused on how well each practice aligned with its category in terms of practical relevance and perceived importance.

5.5.1 Common Sense Practices

Practices classified as *Common Sense* are those expected to be self-evident, intuitive, and widely accepted by practitioners based on their day-to-day experience. Overall, the feedback confirmed the appropriateness of this classification. Practices such as:

- ❑ **CS-01 (Atomic Specification of Test Cases)** and
- ❑ **CS-02 (Complete Independence of Test Cases)**

received predominantly *High* or *Very High* relevance scores. These results validate that the foundational principles of atomicity and test independence are well internalized and valued across different professional backgrounds.

However, some practices within this group received more moderate responses. For instance:

- ❑ **CS-03 (Coverage of Normal and Exceptional Flows)** and
- ❑ **CS-07 (Complex Design for Error Detection)**

showed a broader distribution across relevance levels, indicating that although these are rooted in common sense logic, their practical prioritization may depend on the context or test maturity of the team.

Interestingly, a few respondents noted that overemphasizing certain "common sense" aspects—such as always enforcing atomicity or complexity—could potentially lead to higher maintenance costs or over-engineering.

5.5.2 Literature-Supported Practices

Practices classified as *Literature-Supported* are grounded in empirical research, academic studies, or well-established theoretical models. The responses generally confirmed that these practices are perceived as methodologically robust and relevant, even if they are not as widely internalized as the "common sense" ones.

For example:

- ❑ **LS-03 (Efficient Utilization of Test Code Coverage)** and
- ❑ **LS-05 (Complete Prioritization of Test Case Design)**

were consistently rated with *High* or *Very High* relevance. These practices are often emphasized in formal testing training and frameworks, and the responses suggest practitioners are aware of their structured value.

That said, some practices such as:

❑ **LS-04 (Small Test Code Generation Footprint)** and

❑ **LS-09 (Deterministic Design of Test Results)**

had slightly more variability in the responses. This may reflect the challenge of applying such practices consistently across different projects or the lack of widespread tool support to enforce them effectively.

5.5.3 Summary of Findings

In general, the relevance analysis of the categories confirmed the validity of the original classification. The *Common Sense* practices were broadly endorsed as intuitive and immediately applicable, while the *Literature-Supported* ones were acknowledged for their methodological rigor and long-term value.

However, it also became evident that some practices—although well-categorized—require clearer definitions or contextual examples to ensure consistent understanding and adoption. These insights reinforce the importance of combining empirical validation with theoretical frameworks when designing and promoting testing best practices.

5.6 Practice Prioritization

This step aimed to understand which practices practitioners considered most critical for software testing activities. Participants were asked to rank the 25 code-oriented practices according to their perceived importance, based on their professional experience. To reduce ordering bias, each participant received the practices in a randomized order.

After collecting the rankings from all participants, the average position for each practice was computed and used to order the final ranking. Table 6 presents the prioritized list, from the highest to the lowest importance according to the participants' responses.

The results reveal a clear prioritization pattern among practitioners. Practices such as "Complete Assurance of Test Code Quality" (BP15), "Efficient Utilization of Test Code Coverage" (BP16), and "Atomic Specification of Test Cases" (BP1) were consistently ranked as the most important. These highlight a focus on core principles such as test clarity, coverage, and maintainability.

On the other hand, practices like "Deterministic Design of Test Results" (BP25) and "Responsible Addition of Regression Tests for Fixed Bugs" (BP24) were less frequently prioritized. This does not necessarily indicate they are unimportant but may reflect contextual relevance or a lower perceived impact in daily testing routines.

This prioritized list serves as a useful guide for practitioners and tool designers when selecting and emphasizing software testing practices aligned with professional expectations.

Table 6 – Ranking of Code-Oriented Best Practices Based on Participant Prioritization

Rank	Practice
1	BP15 – Complete Assurance of Test Code Quality
2	BP16 – Efficient Utilization of Test Code Coverage
3	BP1 – Atomic Specification of Test Cases
4	BP2 – Complete Independence of Test Cases
5	BP17 – Complete Prioritization of Test Cases Design
6	BP3 – Coverage of Normal and Exceptional Flows
7	BP4 – Boundary Values Analysis
8	BP6 – Complete Modularity of Test Cases
9	BP5 – Structured Coverage of Testing Process
10	BP7 – Complex Design for Error Detection
11	BP18 – Required Utilization of Missing Tests
12	BP8 – Complete Maintenance of Test Code
13	BP9 – Complete Traceability of Test Cases
14	BP10 – Strict Use of Performance and Security Testing
15	BP19 – Proper Utilization of Test Code Coverage
16	BP11 – Regular Review of Test Cases
17	BP21 – Suitable Utilization of Test Assertions
18	BP12 – Clear Understanding of Test Cases
19	BP13 – Responsible Addition of Test Debugging Comments
20	BP14 – Responsible Addition of Test Code Maintenance
21	BP20 – Small Test Code Generation Footprint
22	BP22 – Complete Avoidance of Test Side Effects
23	BP23 – Suitable Utilization of Test Labels and Categories
24	BP24 – Responsible Addition of Regression Tests for Fixed Bugs
25	BP25 – Deterministic Design of Test Results

5.6.1 Comparison Between Theoretical and Practitioner-Based Rankings

To assess the alignment between theoretical expectations and real-world practitioners' perspectives, we compared the initial ranking of test practices, derived from indicators based on the literature, such as frequency of citation, quality of sources and author impact, with the empirical ranking obtained from the practitioner survey.

Although there is a general trend of consistency between the two rankings, the comparison reveals important divergences that highlight the gap between academic recommendations and practical priorities. Several practices that were theoretically well ranked based on structured literature analysis were deprioritized by practitioners. In contrast, some practices initially placed in lower theoretical positions were the elevated in the practitioner-based ranking.

A notable example of this divergence is Practice BP-17 Add regression tests for fixed bugs, which was placed at the bottom of the theoretical ranking, but emerged as the top-ranked practice in the empirical results. This discrepancy reflects the strong emphasis that

practitioners place on software maintenance, change management, and defect prevention in real-world environments, elements that may be underrepresented or not emphasized adequately in academic sources.

Another example is Practice BP-15 Prioritize test case design, which maintained a relatively stable position in both rankings, indicating convergence in its perceived importance. This suggests that some best practices enjoy both empirical and theoretical validation, reinforcing their status as fundamental principles in software testing.

These observations suggest that, while the theoretical ranking provides a solid conceptual foundation for defining good practices, it should be complemented by practitioner input to ensure that recommendations are relevant, grounded, and responsive to the challenges of real-world software development. The integration of empirical validation, as presented in this thesis, serves not only to improve the legitimacy of the defined practices, but also to promote better alignment between research and practice.

5.7 Tool Support Mapping

As part of the survey's fourth stage, participants were asked to indicate whether they knew of any existing tools that support the implementation or automation of each of the 25 code-oriented practices. The goal of this step was to assess the extent to which the defined good software testing practices are currently supported in the industry by existing tools and to identify any gaps where tool support is lacking.

5.7.1 Method and Data Collection

Each participant received a randomized list of the 25 code-oriented practices and was prompted to answer whether they knew of any tool that supports the practice. In cases where the answer was affirmative, they were invited to name the tool or provide a relevant URL. This open-ended approach allowed for both quantitative (frequency of tool mentions) and qualitative (variation in tool names and types) data collection.

5.7.2 Findings and Normalization of Responses

While the number of responses in this section was smaller compared to other steps, several relevant insights emerged. A total of 23 distinct tool references were collected, although some of them were repeated with different spellings or formats. For instance:

- The tool **Selenium** was mentioned multiple times but with variations such as "selenium", "Selenium", and even as part of a longer description. All were normalized under the canonical name **Selenium**.

- **TestCafe** appeared in two variants: "Test cafe" and "Test cafe & puppeteer". These responses were consolidated under the name **TestCafe**.
- The practice of using test case management tools was reflected in multiple mentions of **TestRail**, and other alternatives like **Xray for Jira** and **Zephyr**.
- Code quality tools such as **SonarQube** and **JaCoCo** were associated with practices involving coverage and test case maintainability.
- For property-based testing, **QuickCheck** and **RSpec** were referenced.
- For behavior-driven development and scenario documentation, **Gherkin** and **pytest markers** were named.
- Interestingly, **GitHub Copilot** was referenced in the context of test code generation or support, showing the rise of generative AI in development workflows.

5.7.3 Analysis and Interpretation

The responses highlight several observations:

- **Tool diversity:** Participants referred to a wide variety of tools, ranging from traditional test automation frameworks (e.g., Selenium, TestCafe) to newer AI-assisted development tools (e.g., GitHub Copilot).
- **Practice-tool alignment:** Many of the mentioned tools align well with specific practices. For example, JaCoCo is clearly associated with practices related to code coverage, while TestRail is a strong match for test case management.
- **Low overall density of responses:** Despite the relevance of the task, not all participants provided input for every practice. This may suggest a lack of awareness of tool-practice mappings, or it could reflect the reality that not all practices are currently supported by tools.
- **Inconsistencies in terminology:** Variants in spelling, formatting, or naming conventions required normalization, which indicates that tool knowledge is often informal and context-dependent among practitioners.

5.7.4 Implications and Observations

The results suggest that while some practices are well-supported by widely known tools (e.g., test automation, coverage, documentation), others may lack direct tool support or at least broad awareness of such tools. Moreover, the fact that respondents tended to name tools differently underscores the importance of offering standardized tool-practice mappings in future versions of the framework.

Additionally, the inclusion of modern and AI-enhanced tools such as GitHub Copilot points to an evolving landscape in which intelligent assistants are becoming more relevant in software testing. This opens a potential path for future integration of AI-based recommendations in tools like TAI-EvalGenTCS.

5.8 Final Considerations

The direct involvement of practitioners was essential to assess the clarity, applicability, and relevance of each practice. Their feedback not only confirmed the practical alignment of many practices, but also highlighted areas that required refinement or further clarification. Practices such as atomic test design, boundary value analysis, and test modularity received strong agreement and high rankings, reaffirming their fundamental role in test case quality. Others generated nuanced responses or lower consensus, prompting reflection on their context of use and formulation.

Throughout the survey process, adjustments were made to enhance the quality of the instrument and improve the reliability of the results. Initial expert reviews and pilot sessions revealed ambiguities in the language of some practices and questions, leading to several refinements before full deployment. In addition, the feedback of the participants during the open text fields revealed valuable insights that helped improve the semantic framework of certain practices and inspired ideas to improve the interface of future evaluations.

Although most practices remained unchanged, minor rewording and clarification were applied to some practices based on recurring observations. Furthermore, randomized display of practices and enforcement of stepwise progression ensured reduction of bias and strengthened the validity of the collected data.

The results presented in this chapter not only reinforce the structure of the proposed set of good software testing practices, but also provide actionable insights for future iterations of the tool and its integration into software development workflows. The next chapter consolidates the main contributions of this research, reflecting on its overall impact, outlining potential areas of future work, and summarizing the key lessons learned during the process.

Chapter 6

Conclusions, Future Work, and Lessons Learned

This chapter presents the main findings and contributions of the thesis, which include the definition and classification of good software testing practices based on practitioner experience, the development of a structured evaluation framework, and the implementation of the TAI-EvalGenTCS tool for AI-based test case evaluation and generation. It also outlines future research directions and discusses lessons learned during the design, experimentation, and validation process with both literature sources and industry professionals. In addition, the results of a practitioner-centered survey are discussed, providing empirical evidence of the clarity, relevance, and applicability of the proposed practices.

6.1 Conclusions

This study highlights the indispensable role of well-designed test cases in software testing, emphasizing their crucial importance in validating software functionality and ensuring overall system reliability. Through a systematic review of the literature, a theoretical foundation was developed that characterizes and classifies good test cases, identifying best practices and essential features that significantly improve the testing process and software quality. This thesis provides a theoretical basis for practitioners to improve their testing strategies.

The findings of this research provide strong evidence in support of the initial hypothesis, demonstrating that validated best practices, combined with AI-powered tools, can systematically enhance the quality of software test cases.

A systematic characterization and classification process was conducted to address what constitutes a good test case. In the first phase, 131 testing practices were extracted

from 103 primary studies. In the second phase, through a detailed analysis, these practices were refined into a final set of 40 best practices, derived from 15 primary studies that provided the most substantial and practice-oriented insights. These practices serve as practical guidelines for test case development and are classified into two main groups: code-oriented practices, which focus on essential aspects of test case design to ensure effectiveness, reliability, and maintainability, and no-code-oriented practices, which encompass broader testing activities such as automation, management, and alignment with business objectives. Furthermore, these practices were further categorized into four groups: common sense, literature-supported, test automation and management, and business-oriented, according to their nature and applicability. This structured classification improves the understanding, adoption, and implementation of test case best practices, providing a practical and evidence-based framework that enhances software testing quality in diverse contexts.

The analysis included ranking various test practices based on key criteria such as the number of references, quality assessment, and the authors' H-index. This ranking process provided insight into the most influential practices, allowing us to define the essential features of a good test case. These rankings serve as a valuable guide for practitioners, enabling them to prioritize and adopt practices widely recognized and supported by the research community. By understanding and implementing these ranked practices, testing teams can refine their approaches, resulting in more effective and reliable test cases.

Furthermore, this study examines the significant impact of testing tools on improving the efficiency, coverage, and reliability of the testing process. According to the authors, integrating testing tools is crucial in enhancing efficiency at various stages of the testing lifecycle, from defect management and test case generation to execution and analysis. However, the scope of this study is limited to the tools discussed by the authors in the systematic review, indicating the need for future research to explore a broader range of tools and their applications. The diverse application of testing tools across different contexts and technologies emphasizes the need for flexibility in their use, even as the industry strives to standardize best practices. Factors such as project size, technologies used, developer experience level, and team maturity can significantly influence the selection and use of testing tools, making it challenging to establish a one-size-fits-all standard.

Despite these valuable contributions, this research highlights the pressing need for more studies to establish additional essential features of test cases in various development contexts. Continued exploration is necessary to adapt these practices to the ever-evolving landscape of software development, ensuring that testing methodologies and tools remain flexible, effective, and relevant.

In addition, the thesis explored the role of testing tools in improving the effectiveness of testing. Although current tools focus on automation, execution, or management, they

often overlook the internal quality of test cases themselves. The findings reveal the need for tools that go beyond execution and support the evaluation and improvement of test design.

TAI-EvalGenTCS introduces an AI-powered approach to evaluate and optimize test cases. Unlike existing solutions that focus on test execution and management, this tool prioritizes test case quality, allowing software teams to improve maintainability and adherence to best practices. Through structured evaluation and AI-powered recommendations, TAI-EvalGenTCS contributes to improving the reliability of software testing efforts.

One of TAI-EvalGenTCS's key contributions is its ability to systematically assess test cases against best practices and automatically refine them using artificial intelligence. Using OpenAI's GPT-4-turbo model, the tool enhances the structure, readability, and modularity of test cases while ensuring adherence to industry-recommended guidelines. The integration of AI-driven insights enables software teams to standardize the quality of test cases, reduce maintenance overhead, and improve the overall robustness of their testing strategies.

Furthermore, the validation process conducted on 16 Java projects demonstrated the tangible benefits of applying AI-based optimizations to the design of the test case. The observed improvements in compliance scores highlight the potential of AI to enhance human expertise in the generation and refinement of test cases. Consistent improvements in test modularity, assertion quality, and boundary condition handling reinforce the importance of incorporating AI into software testing workflows.

The role of best practices in software testing is fundamental and TAI-EvalGenTCS operationalizes these principles in a structured evaluation process. By incorporating both code-oriented and no-code-oriented best practices, the tool not only enforces high-quality test design but also serves as an educational resource for development teams looking to improve their testing methodologies. The ability to quantify the adherence to best practices provides organizations with actionable insights to improve the quality of the test suite over time.

Although the tool offers valuable insights, future work should explore enhancements such as:

- ❑ **Expanding Language Support**: Increasing the range of supported programming languages to cater to diverse software development environments.
- ❑ **Adaptive Scoring Mechanisms**: allowing organizations to customize best-practice scoring criteria based on specific project requirements.
- ❑ **Empirical Validation**: Conduct large-scale experiments to validate the effectiveness of AI-driven test case evaluation in real-world scenarios.
- ❑ **Integration with CI/CD Pipelines**: Enabling seamless integration with continuous integration workflows to provide automated test case evaluations.

By addressing these areas, TAI-EvalGenTCS has the potential to become a standard tool for improving software test quality across industries. Future research will focus on refining its assessment models and expanding its capabilities to ensure widespread adoption and effectiveness.

Furthermore, the practitioner-centered survey (see Chapter 5) provided empirical validation of the proposed practices. The survey confirmed their clarity and relevance, highlighted practical challenges, and allowed the identification of priority practices from an industry perspective. This validation reinforced the theoretical findings and ensured their applicability in real-world contexts.

In general, this research not only defined and validated a structured set of testing practices, but also successfully operationalized them through an AI-powered tool evaluated in experimental and practitioner contexts. These results demonstrate that integrating theoretical foundations, industry insights, and intelligent tools can effectively enhance the quality of software testing practices. This approach addresses the initial gaps identified in the literature and satisfies the objectives set out at the beginning of this research.

6.2 Future Work

Based on the results of this thesis, several opportunities have been identified for future research and tool enhancement.

- ❑ **Contextualization of Best Practices:** Further research can explore how the essential features of good test cases vary between development methodologies (e.g. Agile vs DevOps), domains (e.g. finance, health) and programming languages.
- ❑ **Customizable Evaluation Criteria:** TAI-EvalGenTCS currently uses a fixed set of best practices. Future work could enable configurable scoring mechanisms to align evaluations with project-specific or organizational needs.
- ❑ **Wider Language Support and Tool Integration:** Expanding the compatibility of the tool with other programming languages and integrating it into CI/CD pipelines can extend its applicability and promote adoption in industrial workflows.
- ❑ **Large-Scale Empirical Studies:** While the tool was validated with 16 Java projects, broader evaluations across domains and organizations are needed to generalize the findings.
- ❑ **Operational Profile-Based Test Optimization:** Investigate how operational profiles (POS) can inform the prioritization or generation of test cases, minimizing redundancy and maximizing impact on real user behavior.

6.3 Lessons Learned

Throughout the development of this research and the TAI-EvalGenTCS tool, several technical and methodological lessons were learned:

- ❑ **Tool Development is Non-trivial:** Designing tools capable of collecting runtime data and generating executable test cases requires dealing with challenges related to performance, compatibility, instrumentation, and variability in project architectures.
- ❑ **AI Requires Careful Guidance:** While GPT-4-turbo proved effective in evaluating and generating test cases, the quality of its output depends heavily on prompt engineering and the structure of best practices used as input. Consistent results require a well-structured context and precise instructions.
- ❑ **Practitioner Feedback Adds Value:** The insights collected through the practitioner-centered survey (Chapter 5) proved invaluable in validating and prioritizing the proposed practices, highlighting the importance of involving professionals in the evaluation process.
- ❑ **Best Practices Are a Moving Target:** The interpretation of 'best' practices varies between teams, technologies, and business contexts. Thus, the tool must balance the need to enforce consistent standards and allow flexibility.
- ❑ **Combining Tools May Amplify Impact:** Using multiple specialized tools together (e.g., EvoSuite for generation, TAI-EvalGenTCS for evaluation) can produce more comprehensive and effective test cases than relying on a single tool.

Bibliography

ALEKSANDROVA, E.; DOBRYNINA, N. F. Properties of high-quality test cases. **Modern Technologies and Scientific and Technological Progress**, 2024. Disponível em: <https://doi.org/10.36629/2686-9896-2024-1-107-108>.

ALMOG, D.; HEART, T. What is a test case? revisiting the software test case concept. In: O'CONNOR, R. V. et al. (Ed.). **Software Process Improvement**. Berlin, Heidelberg: Springer, 2009, (Communications in Computer and Information Science, v. 42).

AMRITA; GUPTA, P. Test case prioritization based on requirement. In: _____. **Nombre del Libro**. [S.l.]: Editorial del Libro, 2021. p. 309–314. ISBN 978-981-16-1088-2.

ANWAR, N.; KAR, S. Review paper on various software testing techniques & strategies. **Global Journal of Computer Science and Technology**, p. 43–49, May 2019.

ATIFI, M.; MAMOUNI, A.; MARZAK, A. A comparative study of software testing techniques. In: ABBADI, A. E.; GARBINATO, B. (Ed.). **Networked Systems**. Cham: Springer International Publishing, 2017. p. 373–390. ISBN 978-3-319-59647-1.

AUNG, M. M.; WIN, K. Improving code coverage for structural testing by using parametric inter-procedural slicing. In: **Proceedings of the International Conference on Computer Applications (ICCA)**. [S.l.: s.n.], 2023.

BAQAR, M.; KHANDA, R. The future of software testing: Ai-powered test case generation and validation. **arXiv preprint**, 2024. Disponível em: <https://doi.org/10.48550/arxiv.2409.05808>.

BARRAOOD, S. O.; AL. et. Test case quality factors. **Turkish Journal of Computer and Mathematics Education (TURCOMAT)**, v. 12, n. 3, p. 1683–94, 2021. Disponível em: <https://turcomat.org/index.php/turkbilmat/article/view/990>.

BARRAOOD, S. O.; MOHD, H.; BAHAROM, F. An initial investigation of the effect of quality factors on agile test case quality through experts' review. **Cogent Engineering**, Cogent OA, v. 9, n. 1, p. 2082121, 2022. Disponível em: <https://doi.org/10.1080/23311916.2022.2082121>.

BARóN, A. **Modelo para la definición unificada de la práctica como constructo teórico en ingeniería de software**. Tese (Doutorado) — Universidad Nacional de Colombia, Medellín, 2019.

BLACK, R. **Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing**. Wiley, 2002. (ITPro collection). ISBN 9781601190680. Disponível em: <https://books.google.com.br/books?id=_4NzzQEACAAJ>.

BOWES, D. et al. How good are my tests? In: **2017 IEEE/ACM 8th Workshop on Emerging Trends in Software Metrics (WETSoM)**. [S.l.: s.n.], 2017. p. 9–14.

BRUNETTO, M. et al. On introducing automatic test case generation in practice: A success story and lessons learned. **Journal of Systems and Software**, v. 176, p. 110933, 2021. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121221000303>.

_____. On introducing automatic test case generation in practice: A success story and lessons learned. **Journal of Systems and Software**, v. 176, p. 110933, 03 2021.

CATAL, C. The ten best practices for test case prioritization. In: SKERSYS, T.; BUTLERIS, R.; BUTKIENE, R. (Ed.). **Information and Software Technologies**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 452–459. ISBN 978-3-642-33308-8.

CHANG, S.; QI, Y. **On Tests for Complete Independence of Normal Random Vectors**. 2017. Statistics Theory.

CONG, T.; SHAOYING, L.; ZHENHUA, D. Test case generation from conjunctions of predicates with model checking. **Chinese Journal of Electronics**, 2013.

DALLMEIER, V. et al. Automatically generating test cases for specification mining. **IEEE Transactions on Software Engineering**, 2012.

DEAK, A.; STåLHANE, T. Organization of testing activities in norwegian software companies. In: **2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops**. [S.l.: s.n.], 2013. p. 102–107.

DEBBARMA, M. K. et al. A review and analysis of software complexity metrics in structural testing. **International Journal of Computer and Communication Engineering**, 2012.

DIAS-NETO, A. et al. Toward the characterization of software testing practices in south america: looking at brazil and uruguay. **Software Quality Journal**, v. 25, p. 1145–1183, 2017.

DOBSLAW, F.; FELDT, R.; NETO, F. de O. **Automated Black-Box Boundary Value Detection**. 2022.

DU, H. Test case selection techniques for effective fault localization. In: **Proceedings of the 2023 IEEE International Conference on Computer and Electrical Engineering (IC2ECS)**. [s.n.], 2023. p. 1561–1566. Disponível em: <https://doi.org/10.1109/ic2ecs60824.2023.10493690>.

DYBÅ, T.; YR, T. D. Empirical studies of agile software development: A systematic review. **Information and Software Technology**, Elsevier, v. 50, n. 9, p. 833–859, 2008.

EREIZ, Z. Automating web application testing using katalon studio. In: **Proceedings of the International Scientific Conference on Digital Economy (DIEC)**. Bosnia-Herzegovina: International Business Information Academy, 2019. p. 87–97. Disponível em: <https://www.researchgate.net/publication/343162368_Automating_Web_Application_Testing_Using_Katalon_Studio>.

FABBRI, S. et al. Improvements in the start tool to better support the systematic review process. In: **Proc. of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE'16)**. Limerick, Ireland: [s.n.], 2016.

FEDOSOV, A. et al. Comparative analysis of products for testing software. In: **Software Engineering Perspectives in Intelligent Systems**. Cham: Springer, 2020, (Advances in Intelligent Systems and Computing, v. 1294). p. 242–252. ISBN 978-3-030-63321-9. Disponível em: <https://doi.org/10.1007/978-3-030-63322-6_19>.

FELDERER, M.; SCHIEFERDECKER, I. A taxonomy of risk-based testing. **International Journal on Software Tools for Technology Transfer**, v. 16, p. 559–568, October 2014.

FETA, N.; FITRIA, F. Implementation of concolic unit testing in testing binary search algorithm using jcute. **JITK (Jurnal Ilmu Pengetahuan dan Teknologi Komputer)**, v. 7, n. 2, p. 37–44, Feb. 2022. Disponível em: <https://ejournal.nusamandiri.ac.id/index.php/jitk/article/view/2758>.

FISCHBACH, J. et al. What makes agile test artifacts useful? an activity-based quality model from a practitioners' perspective. **CoRR**, abs/2009.01722, 2020. Disponível em: <https://arxiv.org/abs/2009.01722>.

FONSECA, M. L. Towards understanding students' sensemaking of test case design. **Data and Knowledge Engineering**, 2023.

FRASER, G.; ARCURI, A. Evosuite: Automatic test suite generation for object-oriented software. In: **Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2011. (ESEC/FSE '11), p. 416–419. ISBN 9781450304436. Disponível em: <https://doi.org/10.1145/2025113.2025179>.

GADELHA, G.; RAMALHO, F.; MASSONI, T. Traceability recovery between bug reports and test cases-a mozilla firefox case study. 2021.

GOTA, L.; GOTA, D.; MICLEA, L. Continuous integration in automation testing. In: **2020 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)**. [S.l.: s.n.], 2020. p. 1–6.

GRANERO-GALLEGOS, A.; PHAN, H. P.; NGU, B. H. Advancing the study of levels of best practice pre-service teacher education students from spain: Associations with both positive and negative achievement-related experiences. **PLoS One**, v. 18, n. 6, p. e0287916, June 30 2023.

GUELFI, N.; RIES, B. Selection, evaluation and generation of test cases in an industrial setting: A process and a tool. **Practice And Research Techniques, Testing: Academic & Industrial Conference on**, v. 0, p. 47–51, 08 2008.

Gurock Software. **TestRail: Software de Gestión y Orquestación de Casos de Prueba**. 2025. <https://www.testrail.com/>. Último acceso: 14 de marzo de 2025.

GéLVEZ, H. A. P. **Contribución a la gestión de los procesos de pruebas de software y servicios**. Tese (Doutorado) — Universidad Politécnica de Madrid – Telecomunicacion, 2011.

HAILPERN, B.; SANTHANAM, P. Software debugging, testing, and verification. **IBM Systems Journal**, v. 41, p. 4–12, December 2001.

HAMILTON, L. Testing fate: Tay-sachs disease and the right to be responsible. **Disability & Society**, Routledge, v. 35, n. 6, p. 1026–1027, 2020. Disponível em: <https://doi.org/10.1080/09687599.2019.1646981>.

HASAN, M. M.; BHUIYAN, F. A.; RAHMAN, A. Testing practices for infrastructure as code. In: . [S.l.: s.n.], 2020. p. 7–12.

HOSSAIN, S. B. et al. Measuring and mitigating gaps in structural testing. In: **Proceedings of the International Conference on Software Engineering (ICSE)**. [S.l.: s.n.], 2023.

HUANG, R. et al. Prioritising abstract test cases: an empirical study. **IET Software**, v. 13, n. 4, p. 313–326, 2019. Disponível em: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-sen.2018.5199>.

IBARRA, C. V. et al. Model of best practice representation for any knowledge area by using pre-conceptual schemas. In: **2020 8th International Conference in Software Engineering Research and Innovation (CONISOFT)**. [S.l.: s.n.], 2020. p. 78–85.

IHME, T. et al. Challenges and industry practices for managing software variability in small and medium sized enterprises. **Empirical Software Engineering**, v. 19, n. 4, p. 1144–1168, 2014. ISSN 1573-7616. Disponível em: <https://doi.org/10.1007/s10664-013-9253-0>.

ISLAM, S. et al. Software test estimation tool: Comparable with cocomoii model. In: **2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)**. [S.l.: s.n.], 2016. p. 204–208.

JAHAN, M. S. et al. Software testing practices in it industry of pakistan. In: **Proceedings of the 6th Conference on the Engineering of Computer Based Systems**. New York, NY, USA: Association for Computing Machinery, 2019. (ECBS '19). ISBN 9781450376365. Disponível em: <https://doi.org/10.1145/3352700.3352724>.

JASUJA, V.; SINGH, R. K. Effective approach for code coverage using monte carlo techniques in test case selection. **International Journal of Discrete Mathematics**, v. 2, n. 3, p. 100–106, 2017. Disponível em: <https://doi.org/10.11648/j.dmath.20170203.17>.

JIA, X. **The Role and Importance of Software Testing in Software Quality Management**. 2023. Available at ResearchGate: <https://www.researchgate.net/publication/381409508_The_Role_and_Importance_of_Software_Testing_in_Software_Quality_Management>. Disponível em: <https://doi.org/10.62517/jiem.202303406>.

KAMDE, P. M.; NANDAVADEKAR, V. D.; PAWAR, R. G. Value of test cases in software testing. In: **2006 IEEE International Conference on Management of Innovation and Technology**. [S.l.: s.n.], 2006. v. 2, p. 668–672.

KANG, S.-S.; LARSEN, M. D. Tests of independence with incomplete contingency tables using likelihood functions. **Journal of Data Science**, 2021.

Katalon, Inc. **Katalon Studio: Plataforma de Automatización de Pruebas Potenciada por IA**. 2025. <https://katalon.com/>. Último acceso: 14 de marzo de 2025.

KHATAMI, A.; BRANDT, C.; ZAIDMAN, A. Software quality assurance analytics: Enabling software engineers to reflect on qa practices. In: **Proceedings of the SCAM Conference 2024**. [S.l.: s.n.], 2024. p. 189–200.

KITCHENHAM, B.; CHARTERS, S. Guidelines for performing systematic literature reviews in software engineering. **Technical Report, EBSE**, v. 2, January 2007.

KOCHHAR, P. S.; XIA, X.; LO, D. Practitioners' views on good software testing practices. In: **2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)**. [S.l.: s.n.], 2019. p. 61–70.

KREINOVICH, V.; KOSHELEVA, O.; NGUYEN, H. P. Why h-index. In: **How to Measure and Quantify Research Impact: Innovative Methodologies and Challenges**. [S.l.]: Springer, 2020. p. 53–63.

LI, W.; GALL, F. L.; SPASESKI, N. A survey on model-based testing tools for test case generation. In: . [S.l.: s.n.], 2017. p. 77–89. ISBN 978-3-319-71734-0.

LUCRéDIO, D. et al. Test case quality: An empirical study on belief and evidence. 2023.

_____. **Do good practices lead to more bug-finding effectiveness of test cases? A study based on repository mining**. 2024. Disponível em: <https://ssrn.com/abstract=4798301>.

MAJCHRZAK, T. A. Best practices for the organizational implementation of software testing. In: **2010 43rd Hawaii International Conference on System Sciences**. [S.l.: s.n.], 2010. p. 1–10.

MEIXNER, K. et al. Efficient test case generation from product and process model properties and preconditions. In: **2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)**. [S.l.: s.n.], 2020. v. 1, p. 859–866.

MULLA, F. A. Modern mobile testing tools: A comprehensive guide to quality assurance and automation. **International Journal of Scientific Research in Computer Science, Engineering and Information Technology**, v. 10, n. 6, p. 1577–1584, 2024.

NAHEED, M.; NADEEM, A.; ZAMAN, Q. U. A requirement based approach to test case prioritization for regression testing. In: **2022 19th International Bhurban Conference on Applied Sciences and Technology (IBCAST)**. [S.l.: s.n.], 2022. p. 358–363.

NAIR, A. R. R. C. Crafting effective test cases: Best practices for robust quality assurance. **International Journal For Multidisciplinary Research**, v. 5, n. 6, 2023. Disponível em: <https://doi.org/10.36948/ijfmr.2023.v05i06.25527>.

NINDEL-EDWARDS, J.; STEINKE, G. The development of a thorough test plan in the analysis phase leading to more successful software development projects. **Journal of International Technology and Information Management**, v. 16, January 2007.

PATEL, P. R.; BHATT, C. Structural coverage analysis methods. In: **Book Title**. [S.l.: s.n.], 2018.

PONTILLO, V. Insights into test code quality prediction: Managing machine learning techniques. In: **Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE)**. [s.n.], 2024. Disponível em: <https://doi.org/10.1145/3661167.3661268>.

PractiTest Ltd. **PractiTest: Plataforma de Gestión de Pruebas para Esfuerzos de QA**. 2025. <https://www.practitest.com/>. Último acceso: 14 de marzo de 2025.

QI, Y.; ZHOU, Y. F. Empirical likelihood method for complete independence test on high-dimensional data. **Journal of Statistical Computation and Simulation**, 2022.

QUDDUS, H. A.; SINDHU, M. A. Structural coverage of ltl requirements for learning-based testing. In: **Proceedings of the International Conference on Information Technology (ICIT)**. [S.l.: s.n.], 2022.

QUESADA-LóPEZ, C.; HERNANDEZ-AGüERO, E.; JENKINS, M. Characterization of software testing practices: A replicated survey in costa rica. **Journal of Software Engineering Research and Development**, v. 7, p. 6, 12 2019.

RAMM, A. G. **Boundary values of analytic functions**. 2023.

Ranorex GmbH. **Ranorex Studio: Automatización de Pruebas de GUI con Diseño de Pruebas Inteligente**. 2025. <https://www.ranorex.com/>. Último acceso: 14 de marzo de 2025.

RAO, K. K. et al. Prioritization of test cases in software testing using m2 h2 optimization. **International Journal of Modern Education and Computer Science (IJMECS)**, v. 14, n. 5, p. 56–67, 2022.

RAO, K. K. et al. Prioritization of test cases in software testing using m2h2 optimization. **International Journal of Modern Education and Computer Science**, 2022.

RIBEIRO, V. V.; CRUZES, D. S.; TRAVASSOS, G. H. A perception of the practice of software security and performance verification. In: **2018 25th Australasian Software Engineering Conference (ASWEC)**. [S.l.: s.n.], 2018. p. 71–80.

_____. A perception of the practice of software security and performance verification. 2018.

ROLANDO, A.; VITA, S. Reference case studies and best practices. **Nombre de la Revista**, v. 30, p. 73–91, 2016.

SALTZER, J. H.; WALDEN, D. The origin of the “mit license”. **IEEE Annals of the History of Computing**, v. 42, n. 4, p. 94–98, 2020. Disponível em: <https://doi.org/10.1109/MAHC.2020.3020234>.

SETIANI, N.; FERDIANA, R.; HARTANTO, R. Test case understandability model. **IEEE Access**, 2020.

SmartBear Software. **TestComplete: Automatización de Pruebas Funcionales para Aplicaciones de Escritorio, Web y Móviles**. 2025. <https://smartbear.com/product/testcomplete/>. Último acceso: 14 de marzo de 2025.

SPATH, P. Continuous integration. In: **Pro Jakarta EE 10: Open Source Enterprise Java-based Cloud-native Applications Development**. Berkeley, CA: Apress, 2023. p. 43–59. ISBN 978-1-4842-8214-4. Disponível em: <https://doi.org/10.1007/978-1-4842-8214-4_5>.

SUN, C.; WEI, X.; DONG, Y. Automatic test case generation for simulation training software. In: **Proceedings of the 2018 3rd International Conference on Automation, Mechanical Control and Computational Engineering (AMCCE 2018)**. Atlantis Press, 2018. p. 127–130. ISBN 978-94-6252-508-5. ISSN 2352-5401. Disponível em: <https://doi.org/10.2991/amcce-18.2018.22>.

T., B. C. Complete testing in agile methodology. **International Journal of Scientific and Research Publications**, v. 12, p. 89–92, December 2022.

TAIPALE, O.; KARHU, K.; SMOLANDER, K. Observing software testing practice from the viewpoint of organizations and knowledge management. In: **First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)**. [S.l.: s.n.], 2007. p. 21–30.

TAIPALE, O.; SMOLANDER, K. Improving software testing by observing practice. In: **Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2006. (ISESE ’06), p. 262–271. ISBN 1595932186. Disponível em: <https://doi.org/10.1145/1159733.1159773>.

TIUTIN, C.; VESCAN, A. Test case prioritization based on neural networks classification. In: **Proceedings of the [Conference Name]**. [S.l.: s.n.], 2022.

UTTING, M. et al. Identifying and generating missing tests using machine learning on execution traces. In: **Proceedings of the International Conference on Artificial Intelligence Testing (AITest)**. [S.l.: s.n.], 2020.

VERMA, A.; CHOUDHARY, A.; TIWARI, S. Software test case generation tools and techniques: A review. **International Journal of Mathematical, Engineering and Management Sciences**, v. 8, p. 293–315, 04 2023.

WANG, C. et al. Automatic generation of system test cases from use case specifications. In: **Proceedings of the 2015 International Symposium on Software Testing and Analysis**. New York, NY, USA: Association for Computing Machinery, 2015. (ISSTA 2015), p. 385–396. ISBN 9781450336208. Disponível em: <https://doi.org/10.1145/2771783.2771812>.

WASEEM, M. et al. Design, monitoring, and testing of microservices systems: The practitioners' perspective. **Journal of Systems and Software**, v. 182, p. 111061, 2021. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121221001588>.

WIJAYASIRIWARDHANE, T. K.; WIJAYARATHNA, P. G.; KARUNARATHNA, D. D. An automated tool to generate test cases for performing basis path testing. In: **2011 International Conference on Advances in ICT for Emerging Regions (ICTer)**. [S.l.: s.n.], 2011. p. 95–101.

WINZINGER, S.; WIRTZ, G. Automatic test case generation for serverless applications. In: **2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)**. [S.l.: s.n.], 2022. p. 77–84.

ZAMBONI, A. B. et al. Start uma ferramenta computacional de apoio à revisão sistemática. In: **Brazilian Conference on Software: Theory and Practice - Tools session**. UFBA: [s.n.], 2010.

Appendix

APPENDIX A

Survey Instrument: Complete Question Inventory

This appendix presents the complete survey instrument used in the study *Survey on Best Practices in Software Testing*. The instrument was implemented in a custom-developed web application (not an off-the-shelf platform), purpose-built to support stepwise progression, conditional logic, and structured qualitative feedback.

Survey Overview

- ❏ **Title:** Survey on Best Practices in Software Testing
- ❏ **Institution:** UFSCar (Universidade Federal de São Carlos)
- ❏ **Total Duration:** ~50 minutes
- ❏ **Total Best Practices Evaluated:** 40 practices
- ❏ **Structure:** 5 sequential steps with *no backward navigation*

Step 1 — General Information (Demographic Data)

Estimated Time: 5 minutes

Personal Information Questions

- ❏ **Full Name** (text)

- ❑ **Gender** (Male, Female, Other)
- ❑ **Email** (email field)
- ❑ **Country** (text)
- ❑ **Job Title** (text)
- ❑ **Education Level** (selection)
- ❑ **Age** (numeric)
- ❑ **Years of Experience in Software Testing** (numeric)

Consent Agreement

- ❑ **Term of Free and Informed Consent:** checkbox "I have read and agree to the Term of Free and Informed Consent".

Step 2 — Best Practices Evaluation (Agreement Assessment)

Instructions. Participants evaluated *code-oriented practices* using a 3-point scale: *Agree*, *Neutral*, *Disagree*. Each practice included an optional observation field.

Note. In Step 2, only the **code-oriented** subset was shown. Below we list the complete inventory of 40 practices used across the survey. Each item indicates whether it was **Code-Oriented: Yes/No** (those marked **Yes** were included in Step 2).

Common Sense Category (14 practices)

- ❑ **BP1 — Atomic Specification of Test Cases**
  Category: Common Sense; Domain: Project Management; Use: Independence.
  Description: One aspect per test; tests in a suite must be independent.
  Code-Oriented: Yes.
- ❑ **BP2 — Complete Independence of Test Cases**
  Category: Common Sense; Domain: Project Management; Use: Separation of activities.
  Description: Independence improves scheduling flexibility and reduces maintenance.
  Code-Oriented: Yes.

❑ **BP3 — Coverage of Normal and Exceptional Flows**
Category: Common Sense; Domain: Quality and Maintainability; Use: Coverage.
Description: Include both standard and exceptional flows.
Code-Oriented: Yes.

❑ **BP4 — Boundary Values Analysis**
Category: Common Sense; Domain: Quality and Maintainability; Use: Coverage.
Description: Exercise inputs at the edges of domains.
Code-Oriented: Yes.

❑ **BP5 — Complete Modularity of Test Cases**
Category: Common Sense; Domain: Quality and Maintainability; Use: Documentation.
Description: Modular, well-commented test code as reference documentation.
Code-Oriented: Yes.

❑ **BP6 — Detailed Analysis of Size and Complexity**
Category: Common Sense; Domain: Quality and Maintainability; Use: Size/Complexity.
Description: Prefer small tests for clarity and maintainability.
Code-Oriented: Yes.

❑ **BP7 — Complex Design for Error Detection**
Category: Common Sense; Domain: Quality and Maintainability; Use: Design.
Description: Complex tests may be needed; maintain balance to avoid over-engineering.
Code-Oriented: Yes.

❑ **BP8 — Complete Maintenance of Test Code**
Category: Common Sense; Domain: Quality and Maintainability; Use: Design.
Description: Design tests prioritising maintainability and sound principles.
Code-Oriented: Yes.

❑ **BP9 — Complete Traceability of Test Cases**
Category: Common Sense; Domain: Project Management; Use: Knowledge management.
Description: Maintain links among test code, requirements, and source code.
Code-Oriented: Yes.

❑ **BP10 — Strict Use of Performance and Security Testing**
Category: Common Sense; Domain: Performance and Security; Use: Performance/Security.
Description: Employ appropriate non-functional tests (performance, security, etc.).
Code-Oriented: Yes.

❑ **BP11 — Regular Review of Test Cases**
Category: Common Sense; Domain: Project Management; Use: Review/Update.
Description: Update tests regularly, especially when requirements change.
Code-Oriented: Yes.

❑ **BP12 — Clear Understanding of Test Cases**
Category: Common Sense; Domain: Quality and Maintainability; Use: Clarity.
Description: Tests should be clear, unambiguous, and simple.
Code-Oriented: Yes.

❑ **BP13 — Structured Coverage of Testing Process**
Category: Common Sense; Domain: Quality and Maintainability; Use: Coverage.
Description: Apply suitable integration testing approaches as required.
Code-Oriented: Yes.

❑ **BP14 — Complete Assurance of Test Code Quality**
Category: Common Sense; Domain: Quality and Maintainability; Use: Quality assurance.
Description: Use metrics (e.g., coverage) for unit test quality assurance.
Code-Oriented: Yes.

Literature-Supported Category (11 practices)

❑ **BP15 — Proper Utilization of Code Coverage**
Category: Literature Supported; Domain: Quality and Maintainability; Use: Quality assurance.
Description: Coverage is crucial but not sufficient alone.
Code-Oriented: Yes.

❑ **BP16 — Required Utilization of Missing Tests**
Category: Literature Supported; Domain: Project Management; Use: Quality assurance.
Description: Use coverage to find gaps and guide new tests.
Code-Oriented: Yes.

❑ **BP17 — Efficient Utilization of Code Coverage**
Category: Literature Supported; Domain: Project Management; Use: Quality assurance.
Description: Higher coverage does not guarantee better fault detection.
Code-Oriented: Yes.

❑ **BP18 — Small Generation Footprint**
Category: Literature Supported; Domain: Quality and Maintainability; Use: Size/Complexity.
Description: Each test should execute minimal code.
Code-Oriented: Yes.

❑ **BP19 — Complete Prioritization of Test Cases Design**
Category: Literature Supported; Domain: Quality and Maintainability; Use: Design.
Description: Prioritise design breadth over raw coverage.
Code-Oriented: Yes.

❑ **BP20 — Responsible Addition of Maintenance**
Category: Literature Supported; Domain: Project Management; Use: Review/Update.
Description: Add tests covering fixed bugs to prevent regression.
Code-Oriented: Yes.

❑ **BP21 — Suitable Utilization of Test Assertions**
Category: Literature Supported; Domain: Quality and Maintainability; Use: Design.
Description: Assertions reveal subtle errors by stating expected behaviour.
Code-Oriented: Yes.

❑ **BP22 — Responsible Addition of Debugging Comments**
Category: Literature Supported; Domain: Quality and Maintainability; Use: Design.
Description: Document common errors and possible causes in test code.
Code-Oriented: Yes.

❑ **BP23 — Deterministic Design of Results**
Category: Literature Supported; Domain: Quality and Maintainability; Use: Design.
Description: Avoid flakiness; design for deterministic results.
Code-Oriented: Yes.

❑ **BP24 — Complete Avoidance of Side Effects**
Category: Literature Supported; Domain: Quality and Maintainability; Use: Design.
Description: Tests must be reliable and repeatable.
Code-Oriented: Yes.

❑ **BP25 — Suitable Utilization of Labels and Categories**
Category: Literature Supported; Domain: Project Management; Use: Organisa-

tional structure.
Description: Employ labels/categories to enable selective execution.
Code-Oriented: Yes.

Automation and Test Management Category (8 practices)

❑ **BP26 — Complete Avoidance of Side Effects (Alt.)**
Category: Automation and Test Management; Domain: Quality and Maintainability; Use: Design.
Description: Duplicate of BP24 with alternative numbering.
Code-Oriented: No.

❑ **BP27 — Suitable Utilization of Labels and Categories (Alt.)**
Category: Automation and Test Management; Domain: Project Management; Use: Organisational structure.
Description: Duplicate of BP25 with alternative numbering.
Code-Oriented: No.

❑ **BP28 — Responsible Utilization of Automatic Test Case Generation**
Category: Automation and Test Management; Domain: Tools and Automation; Use: Tools.
Description: Use generation tools responsibly.
Code-Oriented: No.

❑ **BP29 — Responsible Utilization of Management Tools**
Category: Automation and Test Management; Domain: Tools and Automation; Use: Tools.
Description: Use test management tools effectively.
Code-Oriented: No.

❑ **BP30 — Adequate Utilization of Estimation Tools**
Category: Automation and Test Management; Domain: Tools and Automation; Use: Tools.
Description: Use estimation tools for effort and schedule.
Code-Oriented: No.

❑ **BP31 — Adequate Utilization of Coverage Measurement Tools**
Category: Automation and Test Management; Domain: Tools and Automation; Use: Tools.
Description: Use coverage tools to ensure comprehensive testing.
Code-Oriented: No.

❑ **BP32 — Complete Utilization of Continuous Integration Tools**
Category: Automation and Test Management; Domain: Tools and Automation; Use: Tools.
Description: Integrate CI for automated test execution.
Code-Oriented: No.

❑ **BP33 — Adequate Selection of Testing Tools**
Category: Automation and Test Management; Domain: Tools and Automation; Use: Tools.
Description: Select tools based on project requirements.
Code-Oriented: No.

Business-Oriented Category (7 practices)

❑ **BP34 — Aligned Adjustment of Test Organization**
Category: Business-Oriented; Domain: Project Management; Use: Business alignment.
Description: Align testing activities with business orientation.
Code-Oriented: No.

❑ **BP35 — Responsible Utilization of Risk-Based Testing**
Category: Business-Oriented; Domain: Project Management; Use: Risk management.
Description: Prioritise testing by risk levels.
Code-Oriented: No.

❑ **BP36 — Conscious Impact on Test Organization**
Category: Business-Oriented; Domain: Project Management; Use: Organisational structure.
Description: Business orientation influences testing processes.
Code-Oriented: No.

❑ **BP37 — Conscious Impact on Knowledge Management Strategy**
Category: Business-Oriented; Domain: Project Management; Use: Knowledge management.
Description: Business orientation impacts knowledge management.
Code-Oriented: No.

❑ **BP38 — Careful Measurement of Outsourcing Impact**
Category: Business-Oriented; Domain: Project Management; Use: Knowledge management.
Description: Assess outsourcing impact on testing.
Code-Oriented: No.

- ❏ **BP39 — Complete Utilization of Testing Methodology**
  Category: Business-Oriented; Domain: Project Management; Use: Methodology.
  Description: Follow defined testing methodologies.
  Code-Oriented: No.

- ❏ **BP40 — Clear Definition of Test Responsibilities**
  Category: Business-Oriented; Domain: Project Management; Use: Organisational structure.
  Description: Clearly define roles and responsibilities.
  Code-Oriented: No.

Step 3 — Evaluation of Best Practices (Relevance Assessment)

Instructions. Participants evaluated the *same 40 practices* on three dimensions, using a 5-point scale (*Very Low*, *Low*, *Moderate*, *High*, *Very High*):

- ❏ Level of relevance with **Category**
- ❏ Level of relevance with **Domain**
- ❏ Level of relevance with **Use**

Each practice included an optional observation field.

Step 4 — Prioritization of Best Practices (Ranking Assessment)

Instructions. Participants dragged and ordered all 40 practices from highest to lowest priority. Practices were presented in random order and a total order was required.

Step 5 — Tools for Practices (Tool Support Assessment)

Instructions. For each of the first 25 practices (the code-oriented subset), participants answered:

- ❏ *Tool Support Question:* "Do you know any testing tool that supports this practice?"
- ❏ **Response options:** Yes / No
- ❏ If **Yes**, an observation field was available to provide the tool name and/or URL.

Survey Completion

Upon submission of the final step, a **Thank You Page** confirmed completion and expressed gratitude for the participant's feedback.

Repositories and License (Traceability)

Source Code.

- ❏ Backend: <https://github.com/caviva/surveyBP.git>
- ❏ Frontend: <https://github.com/caviva/survey-ufscar.git>

These repositories are also listed in the Appendix/Annexes of this thesis for persistent traceability.

Type of License. The survey system follows an open-access licensing model (MIT License), allowing users to use, modify, and distribute the software with minimal restrictions, fostering transparency, collaboration, and extensibility.

Custom-built System. The survey was conducted using a bespoke web application tailored to this research (stepwise structure, conditional logic, and qualitative feedback support).