INSTITUTO POLITÉCNICO DE BRAGANÇA Escola Superior de Tecnologia e Gestão

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Change Management in Agile Software Development Projects

Kamila Antunes de Souza Neves - 61441

Thesis presented to the School of Technology and Management in the scope of the Master in Informatics.

Supervisors:

José Eduardo Fernandes

Gabriella Castro Barbosa Costa Dalpra

Alinne Cristinne Corrêa Souza

This document does not include the suggestions made by the board.

Bragança

2023-2025

Change Management in Agile Software Development Projects

Kamila Antunes de Souza Neves - 61441

Thesis presented to the School of Technology and Management in the scope of the Master in Informatics.

Supervisors:

José Eduardo Fernandes

Gabriella Castro Barbosa Costa Dalpra

Alinne Cristinne Corrêa Souza

This document does not include the suggestions made by the board.

Bragança

2023-2025

Dedication

I dedicate this work to my parents, who raised me to be free and always gave me the freedom to be who I am.

Acknowledgment

To God, for giving me strength and allowing me to continue this journey.

To myself, for persevering even in the face of difficulties, for paving this path, and for honoring the dreams of the little girl I once was - who believed, dedicated herself, and nave gave up.

To my parents, who never spared any effort to help me get here. For believing in me, supporting my choices, and giving me the freedom to be who I am. Your love and trust allowed me to dream.

To my sister, my friend, who has always been by my side in the moment I needed most.

To my niece, who, even being so young, understood my absence, gave me strength, and was my reason not to give up - and for always welcoming me with a hug and a smile at every airport throughout this journey.

To my friend, especially Dhiego, Rita e André, who stood by me, supported me, and made this journey lighter and more meaningful. And also to all the other friend and people I met along the way, who in some way were part of this path.

To my advisors, Jef, Gabriella, and Alinne, for their support, guidance, and for walking alongside me with patience and dedication throughout the entire process. A special thanks to professor Gabriella for being present during the journey, for the conversations, and for the constant encouragement.

Finally, I thank IPB for providing the structure and for forstering and academic environment of learning, and UTFPR-DV for the opportunity.

To all of you, my sincere thank you.

Abstract

This study proposes a structured approach to change management in software development projects using agile methodologies. Based on a Systematic Literature Review and an empirical survey with industry professionals, the main causes, impacts, and practices related to changes in agile projects were identified. As a result, a practical guide was developed consisting of three checklists - scope definition, requirements gathering, and change management - and two change request templates (complete and simplified). These artifacts were integrated into the Scrum lifecycle, considering agile roles and appropriate application moments to ensure traceability, collaboration, and informed decision-making. The study provides practical tools to support development teams in mitigation risks, improving communication, and increasing the effectiveness of change adoption in agile environments.

Keywords: change management, agile methodologies, scrum, software engineering, requirements, scope.

Resumo

Este trabalho propõe uma abordagem estruturada para a gestão de mudanças em projetos de desenvolvimento de software que utilizam metodologias ágeis. A partir de uma Revisão Sistemática da Literatura e de uma pesquisa empírica com profissionais da área, foram identificadas as principais causas, impactos e práticas relacionadas a mudanças em projetos ágeis. Como resultado, foi desenvolvido um guia prático composto por três checklists - definição de escopo, levantamento de requisitos e gestão de mudanças - e dois modelos de requisição de mudanças (completo e simplificado). Esses artefatos foram integrados ao ciclo de vida do Scrum, considerando os papéis ágeis e os momentos ideias de aplicação, visando garantir rastreabilidade, colaboração e tomada de decisão embasada. O estudo contribui com ferramentas práticas para apoiar equipes de desenvolvimento na mitigação de riscos, melhorias da comunicação e aumento na eficácia de mudanças em ambientes ágeis.

Palavras-chave: gestão de mudanças, metdologias ágeis, scrum, engenharia de software, requisitos, escopo.

Contents

List of Figures

Acronyms

ACT Academic Training.

AM Agile Methodology.

ARCM Agile Requirement Change Management.

CVI Content Validity Index.

Dev Developer.

EC Exclusion Criteria.

ECM Experience Change Management.

EPM Experience Project Management.

ESTiG School of Technology and Management.

FR Functional Requirements.

GQ Guiding Questions.

GQM Goal-Question-Metric.

GSD Global Software Development.

HE Higher Education.

IC Inclusion Criteria.

IHE Incomplete Higher Education.

IPB Polytechnic Institute of Bragança.

NFR Non-Functional Requirements.

Org. Size Organization Size.

PICO Population, Intervention, Comparison and Outcomes.

PM Product Manager.

PO Product Owner.

QA Quality Assurance.

RA Requirements Analytic.

RE Requirements Engineering.

RQ Research Questions.

SLC Software Lifecycle.

SLR Systematic Literature Review.

SM Scrum Master.

TDD Test-Driven Development.

TL Teach Leader.

UTFPR-DV Federal University of Technology - Paraná, Dois Vizinhos campus.

XP Extreme Programming.

Chapter 1

Introduction

This chapter introduces the motivation behind this dissertation and presents the research problem, hypothesis, and guiding questions. It also describes the study's goals, outlines the adopted methodology, and highlights the expected contributions of the research.

1.1 Motivation

Agile methodologies comprise a set of techniques and practices designed to manage projects through iterative cycles and flexible development processes. Their primary aim is to increase efficiency, adaptability, and delivery speed - characteristics that align with current market demands for immediate results in environments marked by uncertainty and constant change. Agile methodologies stand out for their short, product-focused iterations, collaborative decision-making, continuous integration of functionalities, and rapid incorporation of changes [1] [2].

The Agile Manifesto [2], developed in 2001, established principles that promote innovative practices for system management and development. Focusing on customer satisfaction and continuous interaction between the project team and the client, the Manifesto [2] proposes:

- Valuing individuals and interactions over processes and tools;

- Prioritizing working software over comprehensive documentation;
- Favoring customer collaboration over contract negotiation;
- Adapting to changes rather than rigidly following a plan.

The agile movement emerged as a response to traditional software development life cycle models, such as the Waterfall, which were characterized by rigidity, bureaucracy, and an excessive focus on documentation. While the Agile Manifesto does not disregard the importance of documentation and contracts, it considers them secondary to delivering value and project progress. A distinctive feature of agile methodologies is the prioritization of solving problems dynamically and efficiently, thereby reducing time spent on documentation [3].

In parallel, Change Management plays a fundamental role in software project management and in enabling sustainable transformations within organizations. It is understood as the structured application of methods, tools, and processes to guide individuals and teams through the transition from a current state to a desired future state. According to [4], effective change management ensures that individuals can adopt and internalize organizational changes, aligning people, processes, and technology toward strategic goals.

A structured change process often incorporates individual-focused strategies. One widely recognized framework is the ADKAR model, which outlines five key building blocks necessary for successful individual change [4].

1. **Awareness** of the need for change;
2. **Desire** to support and participate in the change;
3. **Knowledge** on how to change;
4. **Ability** to implement new skills and behaviors;
5. **Reinforcement** to sustain the change.

The ADKAR model emphasizes that organizational change occurs only when individual change is successful, reinforcing the idea that managing the human side of transformation is critical as managing its technical aspects.

In the context of traditional software development methodologies - such as the Waterfall model, the V-Model, and the Rational Unified Process (RUP) - changes were rarely accepted due to their reliance on detailed upfront planning, rigid sequential phases, and extensive documentation. These approaches typically assume that requirements are fully known and stable at the beginning of the project, making late-stage changes costly and disruptive [5][6]. For instance, the Waterfall model follows a linear progression through phases such as requirements analysis, system design, implementation, testing, and maintenance, with minimal feedback loops [7]. Similarly, the V-Model emphasizes strict validation and verification phases tied to the corresponding development stages, making it difficult to accommodate changes once a phase is completed. The RUP, while more iterative, still maintains formal documentation and structured workflows, limiting its flexibility compared to agile approaches [8].

In contrast, agile methods embrace change as a central principle. Changes in client needs are expected and even encouraged, as illustrated by the Agile Manifesto's precept: *"Welcome changing requirements, even late in development"* [2]. Agile methodologies prioritize adaptability, iterative delivery, and continuous stakeholder involvement, enabling development teams to respond more effectively to evolving requirements and business goals.

1.2 Problem, Hypothesis, and Research Questions

1.2.1 Problem

The software development environment is characterized by constant changes in customer needs, emerging technologies, and market uncertainties. Agile methodologies have emerged as a response to these demands, prioritizing adaptability and flexibility. However, when

poorly managed, these changes can have adverse impacts on cost, schedule, product quality, and stakeholder satisfaction.

Although the literature provides studies on change management in software development [3], [9]–[11], there remains a gap between academic theory and practical application in organizational contexts. This disparity limits the effective use of strategies that could mitigate the negative impacts of change in agile projects.

1.2.2 Hypothesis

The adoption of agile methodologies facilitates the management, acceptance, and implementation of changes in software development projects, reducing negative impacts on cost, schedule, and product quality. However, the effectiveness of this approach relies on the consistent application of clear communication practices, active stakeholder involvement, and careful impact assessment.

1.2.3 Research Questions

To address the problem and test the proposed hypothesis, the following research questions were formulated:

- **RQ1**: What types of changes have the greatest impact on software development projects?
- **RQ2**: What are the impacts of changes during software development projects?
- **RQ3**: Does the use of agile methodologies help manage, accept, and implement these changes in a way that does not significantly impact the project?
- **RQ4**: How are changes managed and communicated to the team?

1.3 Goals

This study aims to enhance the understanding of change management in software development projects that use agile methodologies. It seeks to bridge the gap between theoretical approaches and practical experience by analyzing how changes are addressed in the literature and how they manifest in real-world software projects.

Given that changes can occur at any stage of the software lifecycle, it is essential to understand how they are identified, communicated, and managed. To support this investigation, a systematic literature review and an empirical survey were conducted. The results obtained contributed to the formulation and refinement of the approach proposed in this thesis.

The specific goals of this research include:

1. **Explore Change Management Approaches**: Investigate and compare various change management strategies in software projects, including those aligned with agile methodologies, configuration management models, and formal change control processes.

2. **Analyze Types of Changes in Software Projects**: Identify and categorize common types of changes, such as modifications in client requirements, adoption of new technologies, team restructuring, and other factors that significantly impact project execution.

3. **Identify Challenges in Change Management**: Examine the primary difficulties organizations face in managing changes, including resistance to change, communication breakdowns, and obstacles in operationalizing change.

4. **Evaluate the Impact of Changes**: Assess the effects of changes on project performance, focusing on critical success criteria such as quality, schedule adherence, and cost control.

5. **Propose Effective Strategies**: Develop and suggest actionable strategies that enable software teams to manage changes efficiently while minimizing negative impacts

on project outcomes.

6. **Contribute to Knowledge on Change Management**: Provide a practical and research-based framework that supports organizations in improving their change management capabilities, particularly within agile environments.

These goals converge towards a central goal: to generate evidence-based insights and propose practices that organizations can adopt to improve their software project success rates, especially in volatile and fast-changing environments.

1.4 Research Methodology

The process study consisted of two main stages: evidence collection through a Systematic Literature Review (SLR) and a survey, followed by analysis and interpretation of the results. In addition, the checklists developed based on the findings were submitted for expert validation. This validation involved professionals with substantial experience in agile software development and change management, who evaluate the relevance, clarity, and adequacy of the checklist items.

1.4.1 Evidence Collection

Systematic Literature Review

The SLR was conducted to provide an overview of the primary types of change requests in agile software development projects, as well as their impacts. The analysis of 18 selected primary studies revealed a significant gap in understanding of how these changes affect the project lifecycle - positively or negatively. Moreover, most studies focused primarily on requirement-related changes, with other dimensions remaining underexplored.

Survey

Following the SLR, a survey was conducted with the objectives of:

- To identify the most frequent change requests in software projects.
- To evaluate the impact of these changes on the process and project quality.
- To understand how these changes are managed and communicated throughout the project lifecycle.

1.5 Main Contributions

This dissertation contributes to both academic research and industry practice by:

- Providing a structured overview of how change management is addressed in agile software development projects;
- Identifying and categorizing the most common types of changes and their impacts;
- Revealing gaps between theoretical frameworks and real-world practices through empirical evidence;
- Proposing actionable strategies to improve the way organizations manage changes in agile contexts.

1.6 Text Structure

In addition to this introductory chapter, this dissertation is organized into the following chapters:

- **Chapter 2 - Background**: Presents the foundational concepts related to software project management, agile methodologies, change management, and the types and impacts of changes that occur during software development.
- **Chapter 3 - Literature Review**: Describes the systematic literature review conducted, including the research protocol, selection criteria, and key findings regarding change management in agile software projects.

- **Chapter 4 - Survey**: Details the design, execution, and results of the survey conducted with professionals working in software development across diverse domains, including education, finance, manufacturing, industry, and technology. The objective was to understand how changes are perceived, communicated, and managed in practice.

- **Chapter 5 - Proposed Approach**: Presents the practical contribution of this research in the form of checklists and templates to support change management, and discusses the alignment with agile roles and workflows.

- **Chapter 6 - Final Considerations**: Summarizes the main contributions and limitations of the research, and suggests directions for future investigations, including improvements to the proposed approach and broader validations.

- **Bibliography and Appendices**: Includes all references used throughout the dissertation and the Appendices containing supporting materials, such as the complete checklists and survey instruments.

Chapter 2

Background

This chapter provides the theoretical and contextual background necessary for understanding this dissertation, including key concepts and relevant approaches that support the development of this work.

2.1 Agile Methodologies

Agile methodologies emerged in the 1990s as a evolution of iterative and incremental development practices, aiming to better handle changing requirements and customer collaboration, which were less emphasized in traditional approaches like the Waterfall model. Initial agile frameworks included Scrum, Crystal, and Extreme Programming (XP), which introduced more iterative and collaborative processes. These ideas were later consolidated in the Agile Manifesto in 20001, which formalized the values and principles of agile software development [2].

Agile methodologies represent a set of practices designed to increase project efficiency, adaptability, and value delivery. They are characterized by short, iterative development cycles (sprints), where teams divide complex projects into smaller and manageable increments. This enables continuous development, frequent feedback, and rapid response to change - an essential capability in fast-paced, uncertain environments [12].

Key characteristics of agile methodologies include [2]:

1. **Short and Iterative Cycles**: Projects are executed in sprints, enabling incremental and continuous delivery. Feedback from stakeholders is incorporated at the end of each cycle to refine the solution and adjust priorities.

2. **Collaboration and Communication**: Frequent interaction between development teams, clients, and stakeholders is emphasized. Agile favors face-to-face communication and close collaboration to enhance transparency and shared responsibility.

3. **Flexibility and Adaptability**: Agile welcomes changes, even in later stages of development. This flexibility allows teams to adjust plans and deliver value that aligns with evolving requirements.

4. **Continuous Value Delivery**: Each iteration delivers a working product increment, allowing stakeholders to see tangible results throughout the project lifecycle.

5. **Focus on Simplicity**: Agile promotes simple and elegant solutions. Practices such as code refactoring, automated testing, and Test-Driven Development (TDD) help maintain high code quality and adaptability.

The Agile Manifesto also outlines four values that guide practices [2], including:

- Satisfying the customer through early and continuous delivery of valuable software;
- Welcoming changing requirements, even later in development;
- Promoting face-to-face communication as the most effective way to convey information;
- Encouraging continuous collaboration between business stakeholders and developers.

2.2 Change Management

Change Management is defined as "the application of a structured process and a set of tools to lead the human side of change, aiming to achieve a desired outcome" [1]. This

process typically includes essential steps such as planning, implementation, monitoring, and reinforcement. Its objective is to ensure that changes are efficiently integrated, disruptions are minimized, and intended results are achieved. Also referred to as Change Enablement, this discipline ensures that changes to services, systems, or processes are thoroughly evaluated, planned, and approved before implementation. As outlined in [13], proper change control reduces risks and supports safer and more organized transitions.

According to the Guide to the Software Engineering Body of Knowledge (SWEBOK) [14], effective change management relies on well-defined processes that include the evaluation, approval, implementation, and controlled review of all change requests. Each request must be logged and classified to maintain traceability and support appropriate prioritization.

As defined in ITIL 4 Change Enablement [13], change management is guided by four fundamental principles:

1. **Understand**: Before implementing any change, it is essential to develop a comprehensive understanding of the proposal, its motivation, and its potential impacts. This involves:

    - Analyzing the current state;
    - Identifying the underlying need for change;
    - Assessing how the change will affect processes, people, and the organization as a whole.

2. **Plan**: Once the need for change is clearly understood, careful planning is critical to ensure success. This step includes:

    - Establishing clear objectives.
    - Defining the scope and boundaries of the change.
    - Identifying potential risks and developing mitigation strategies.
    - Creating a detailed roadmap to guide implementation efforts.

3. **Implement**: At this stage, the plan is executed in a controlled and coordinated manner. Key actions include:

    - Monitoring progress and performance indicators;
    - Adjusting as needed in response to unforeseen challenges;
    - Ensuring leadership engagement to drive the successful execution of the change.

4. **Communicate**: Effective communication is the cornerstone of successful change management. It must be continuous and consistent, involving:

    - Providing clear and timely information to all stakeholders;
    - Sharing regular updates on progress and outcomes;
    - Promoting transparency to reduce resistance and build support.

2.3 Change Management in the Context of Agile and Traditional Methodologies

While the concept of change management remains consistent across development modes, the way changes are handled throughout the project lifecycle differs significantly between traditional and agile approaches.

In traditional software development lifecycle models - such as the Waterfall - change are typically treated with caution, as they may introduce risks to project scope, schedule, or quality. These models rely on detailed, sequential planning, where modifications introduced after the early stage (e.g., analysis and design) can result in high costs, delays, and disruption to tightly defined schedules [15], [16]. In such environments, change management is typically reactive, requiring formal analysis, documentation, and multiple approval layers before any modification can be implemented [4] [17].

In contrast, agile methodologies not only accept but actively embrace it as a driver of value. According to the Agile Manifesto: "*Welcome changing requirements, even late in*

development. Agile processes harness change for the customer's competitive advantage" [2] [18]. This principle reflects the core agile commitment to rapidly evolving stakeholder needs and market dynamics. By breaking work into short, incremental cycles (sprints), agile teams continuously revisit and refine requirements, incorporate ongoing feedback, and adjust the project scope with minimal impact on cost or schedule [19] [20].

Agile change management is therefore not an isolated activity but a continuous and integrated part of the development process-promoting responsiveness, collaboration, and customer satisfaction.

2.3.1 Key Differences in Change Management

1. **Proactive vs Reactive Approach**

    - In the traditional model, changes are often seen as disruptions to the original plan. Accepting alterations typically requires formal approval processes, with high costs associated with changes made in later stages of development [16] [17].
    - In the agile models, changes are are anticipated from the beginning. Teams are expected to adapt continuously throughout the development process [11] [20].

2. **Continuous Integration of Changes**

    - Traditional models follow a linear structure where requirements are defined early and subsequent change may lead to significant rework and delays [4] [17].
    - Agile promotes the continuous integration of changes in each sprint, reducing rework and maximizing client value [11] [18].

3. **Constant Feedback**

    - In the traditional models, feedback tends to be limited and is usually collected at the end of the process lifecycle [16] [19].

- Agile encourages feedback from clients and stakeholders during every iteration, enabling continuous alignment with expectations [18] [20].

4. **Flexible Scope and Planning**

   - Traditional development relies on detailed upfront planning. Change introduced later may compromise the budget or project timeline [11] [17].
   - Agile methodologies use a prioritized backlog, allowing teams to incorporate new demands or adjustments dynamically, without compromising overall progress [19] [21].

2.3.2 The Competitive Advantage of Agile Methodologies

By treating change as an opportunity rather than a threat, agile methodologies create a competitive advantage for organizations. The ability to respond quickly to client needs - even in later stages of development - not only increases stakeholder satisfaction but also enables the delivery of solutions better aligned with market demands [2] [11] [19].

While traditional models focus on controlling change, agile methods prioritize adapting to it, using changes as a *catalyst for innovation and continuous improvement* [15] [18] [21]. This mindset, consistent with the values of the Agile Manifesto, positions change management as a strategic differentiator in agile software projects.

2.4 Change Types in Software Development Projects

This section explores the main categories of changes typically encountered in software development projects, highlighting their definitions.

2.4.1 System Requirements

The requirements of a system are defined as "descriptions of the services the system must provide, as well as the constraints under which it must operate" [6]. In general, requirements specify the system's functional and operational expectations and can be divided into two main categories: functional requirements (FR) and non-functional requirements (NFR).

Functional Requirements (FR)

Functional requirements refer to the behaviors, functionalities, and services that the system must offer. They describe, for example, how the system should respond to specific inputs, how it should process information, and how it should behave in certain situations. According to [14], these requirements outline the system's scope, necessary business functions, and structured data involved.

These requirements are usually obtained through elicitation activities with stakeholders (end users, client, system owners) and must undergo careful analysis to avoid ambiguities, redundancies, and contradictions. Although often directly associated with functionalities, functional requirements represent the system's functional need - i.e., expected actions or behaviors that may be executed by users or by the system itself.

The process of defining functional requirements is composed of four main phases: Elicitation, Analysis, Specification, and Validation [14].

Non-Functional Requirements (NFR)

Non-functional requirements refer to the constraints and qualities that the system must possess during its development, operation, and maintenance. According to [6], they are also known as complementary requirements or constraint statements, as they impose conditions that limit or qualify the functional requirements.

These requirements do not describe specific system behaviors but rather desirable

characteristics such as usability, reusability, reliability, performance, efficiency, maintainability [22], [23], [24]. They may also include technical constraints, such as mandatory use of a specific programming language, platform, or security standard.

These attributes are essential to ensure that the system is acceptable not only in terms of "what it does," but also "how it does it". Although they do not represent direct functionalities, they are fundamental to ensure the software adequately fulfills its intended purpose and delivers value to the user [25].

2.4.2 Technology in Software Development Projects

In the context of software engineering, the term *software technology* refers to the set of tools, programming languages, frameworks, libraries, development platforms, databases, runtime environments, and automated processes used to plan, design, build, test, deploy, and maintain software systems.

According to the [14], technology in software projects includes both technical artifacts and development support environments, such as version control systems, continuous integration tools, test automation, cloud platforms, and code repositories. These technologies act as enablers for project activities, directly influencing the productivity, quality, and flexibility of deliverables.

As stated by [6], the selection and evolution of software technologies have a significant impact on the project, especially concerning the system architecture, the quality of the final product, and the ability to adapt to changes. Technological changes - such as the adoption of a new framework or migration to a different infrastructure - may present opportunities to improve performance and scalability, but also introduce challenges related to learning curves, compatibility, and rework.

2.4.3 Team in Software Development Projects

The project team in software development consists of professionals responsible for planning, analyzing, designing, building, testing, delivering, and maintaining technological

solutions. This team may include, among other roles: requirements analysts, developers, software architects, testers (QA), project managers, DevOps engineers, interface designers, and both technical and business stakeholders.

According to the [26], the project team is a fundamental resource for the success of the endeavor and must possess technical competencies, interpersonal skills, and the ability to adapt to change. The guide emphasizes that team development is an ongoing process aimed at improving both individual and collective performance, fostering collaboration and effective decision-making.

As mentioned by [6], team changes - such as the addition or departure of members, role reallocation, or replacements by new professionals - are common in long-term projects and can significantly impact work continuity, the retention of accumulated knowledge, and development pace. According to the author, staff turnover is a critical risk factor that requires special attention from project managers.

2.5 Change Impacts in Software Development Projects

Changes in software development projects can significantly affect key project dimensions such as cost, schedule, scope, workflow, and quality, besides the generation of rework. Understanding these impacts is essential for effective change management and informed decision-making throughout the project lifecycle. This section explores these possible impacts, highlighting their definitions.

2.5.1 Cost in Software Projects

Estimating and managing costs is a fundamental aspect of software project management. Cost represents the total financial effort required to complete a project, including labor, tools, infrastructure, licenses, training, maintenance, and other operational expenses. In software projects, however, cost estimation is notably challenging due to the intangible nature of the product, frequent changes in requirements, and variability in team experience and productivity. According to Pressman [27], software cost and effort estimation will

never be an exact science, as too many variables - human, technical, environmental, and political - can influence the outcome. Nevertheless, estimation can be approached systematically to reduce uncertainty and achieve an acceptable level of risk. Pressman [27] suggests four main strategies for improving accuracy in software cost estimation:

1. Postponing the estimation until the project is better defined;
2. Basing estimates on data from similar completed projects;
3. Applying decomposition techniques to break down the project into smaller components;
4. Using empirical models such as COCOMO to support estimation through mathematical formulas.

Ideally, these techniques should be used in combination to validate one another and increase reliability. Decomposition-based methods adopt a "divide and conquer" approach, where the project is broken into functional and technical components. Empirical models, on the other hand, use historical data to produce effort and cost estimates using predefined parameters. As highlighted by [28], cost is influenced by multiple project-specific factors, including:

- The size and complexity of the software;
- The level of expertise and experience of the development team;
- The effectiveness of the tools and technologies used;
- The degree of software reuse possible from previous systems.

In agile environments, cost estimation is further complicated by the iterative nature of the process and the lack of fixed requirements. While agile values prioritize individuals and interactions, working software, and adaptability [2], accurate cost forecasting remains essential. Estimation tools in agile projects typically rely on historical data and model-based techniques, such as COCOMO, SLIM, SEER-SEM, and others, which use system

size (e.g., story points, function points) as key input parameters. Estimation approaches are commonly categorized into:

- **Regression-based model** - rely on statistical analysis of historical data;
- **Expert judgment** - rely on human experience and past project analogies;
- **Learning-oriented model** - adapt over time based on new data and feedback;
- **Bayesian composite model** - combine multiple sources of evidence for probabilistic estimation.

2.5.2 Schedule

In software project management, the schedule refers to the planned distribution of effort across the project's timeline, associating estimated work with specific software engineering tasks. As described by [25], scheduling begins with the creation of a macroscopic schedule, which outlines the main phases and deliverables of the project. As development progresses, this is refined into a detailed schedule, where each task is clearly identified, sequenced, and assigned to specific team members.

Effective scheduling is guided by fundamental principles:

- Decomposition of the project into manageable tasks and activities;
- Identification of tasks interdependencies, recognizing which tasks must be executed sequentially and which can occur in parallel;
- Time allocation, estimating the duration required for each activity;
- Effort allocation, ensuring team capacity is not exceeded at any point;
- Assignment of responsibilities, clearly defining deliverables for each scheduled task;
- Definition of milestones, used as a checkpoint to assess progress and verify the quality of deliverables.

2.5.3 Scope in Software Projects

In software project management, scope refers to the boundaries of what will be delivered. It defines the functions, features, constraints, interfaces, and performance expectations of the software to be developed. According to [27], a well-defined scope must be unambiguous and understandable both at the managerial and technical levels. This clarity is essential to establish realistic planning, avoid misunderstandings, and control changes during the development process.

The software scope describes several key elements:

- The functionalities and characteristics to be provided to users;
- The input and output data;
- The content presented to users during usage;
- The performance requirements, constraints, and reliability expectations that define and limit system behavior.

The first formal activity in a software project is to define its scope. This involves answering fundamental questions such as:

- **Context**: How does the software fit into a larger system, product, or business environment? What constraints arise from that context?
- **Information objectives**: What data objects (visible to the client) are generated as outputs? What input data is required?
- **Function and performance**: What transformations does the software perform on the input data? Are there any special performance or quality characteristics that need to be addressed?

Once the scope is identified, it is necessary to evaluate whether the project is feasible. As [29] highlight, not everything that is imaginable is necessarily achievable - even in software, which might appear highly malleable from the outside.

2.5.4 Rework in Software Projects

Rework is one of the main factors that compromise productivity and efficiency in software projects. According to [26], rework is defined as: *"Action taken to bring a defective or nonconforming component into compliance with requirements or specifications"*.

In other words, rework is required whenever a project artifact - such as code, documentation, or specifications - needs to be revised or redone because it does not meet previously established criteria.

Although it is a common activity, many organizations underestimate the impact of rework, especially when there is no strict quality control and changes are not well managed. According to [30], in software projects, the cost of rework can exceed the cost of all other activities combined - particularly when defects are discovered in the later stages of the software development lifecycle. The later an error is identified, the more expensive it is to fix.

Moreover, as requirements change - which is especially common in agile projects or volatile environments - rework becomes inevitable. Changes to requirements directly impact previously developed artifacts, requiring updates to specifications, design, source code, and testing. This consumes time and resources and may compromise the project's stability.

The [6] reinforces that unplanned or poorly documented changes often lead to unnecessary rework, resulting in loss of productivity, delays, and increased costs. Therefore, effective requirements management and continuous validation throughout development are fundamental practices for minimizing rework and ensuring more consistent deliveries.

2.5.5 Software Quality

Software quality is a central concept in software engineering and encompasses various aspects related to the system's ability to meet established requirements and stakeholder expectations. According to the [31], software quality is defined as: *"The degree to which a software product conforms to established requirements and satisfies stated and implied*

needs under specified conditions".

This definition is complemented by [6], who asserts that software quality is also related to the extent to which established requirements accurately represent the needs, desires, and expectations of the stakeholders involved in the project. In other words, it is not enough for the software to fulfill documented requirements - those requirements must faithfully reflect what users need.

Therefore, software quality goes beyond the correct implementation of functionalities. It is also strongly associated with non-functional attributes such as performance, usability, security, and reliability, among others. These attributes are critical to the user experience and the robustness of the solution, especially in environments with complex or critical requirements.

2.5.6 Workflow

The workflow in software development can be defined as the structured sequence of activities, tasks, and interactions between people, processes, and tools that are necessary to achieve a specific objective within the project - such as delivering a functional requirement, fixing a defect, or integrating a change.

According to [32], the workflow consists of a logical and controlled order of actions, which can be automated (using management and execution tools) or manual (depending on human coordination). In software projects, it typically includes activities such as requirements gathering, design, coding, testing, deployment, and maintenance, connecting the different roles within the team.

In the context of change management, the workflow also acts as a control mechanism, since each change - wether related to requirements, technology, or team composition - must follow a formal path within the project, such as approval, planning, implementation, and validation. This is where frameworks like ITIL and practices like DevOps add value by establishing formal channels for the safe and continuous introduction of changes.

In agile approaches, the workflow is iterative and incremental, with short delivery

cycles (sprints), daily alignment meetings (dailies), and visual boards (such as Kanban) enabling adaptive workflows essential to eliminating waste, avoid rework, and maximize the value delivered to the customer.

2.6 Related Work

To develop an initial understanding of how agile methodologies handle changes and how organizations apply change management practices, a literature review was conducted considering the PICO framework [33]. This literature review is detailed in Section 3, including its protocol and findings. These studies were extracted from the review as it is considered essential to compose this section on related work. Below, we present a comparative analysis of the studies, which enables the identification of different approaches, challenges, and best practices discussed in the literature.

Fogarty [34] highlights often overlooked factors in agile organizations, such rework, maintainability issues, staff turnover, and the associated costs. These elements can significantly impact project outcomes, especially in agile environments where frequent changes are expected. When managers fail to effectively oversee these dynamics, organizations may incur hidden, high costs. Although this thesis focuses on the impact and management of change in agile contexts, Fogarty's finding underscore the financial and operational consequences of inadequate change planning and control.

Alawairdhi [35] demonstrates that agile methodologies can function as practical tools for change management, particularly in small to medium-sized projects. The study shows that continuous user involvement and improved product visibility contribute to more effective change handling, proving clear benefits for the higher education sector in Saudi Arabia. While this thesis also investigate changes in agile environment, Alawairdhi's work differs by specifically addressing how change is accepted and managed across all development phases, with emphasis on communication, impact evaluation, and team response.

Aizaz's [9] identifies scope creep as a leading cause of failure in traditional software projects, arguing that effective scope change management is crucial for project success.

The study focuses on Global Software Development (GSD) and proposes a conceptual model to manage these changes. While Aizaz centers the discussion on geographically distributed teams, this thesis adopts a broader perspective by exploring how change occurs and is managed throughout various phases of agile development, analyzing it impact on quality, time, and cost.

Alburquerque's [36] proposes a structured process for Agile Requirement Change Management (ARCM) and identifies supporting practices to improve how teams handle evolving requirements. Building on this, our work broadens the scope by analyzing not only how requirements change, but also how teams communicate and evaluate the broader impacts of various types of changes throughout the agile development cycle.

Raza [37] explores change management across different agile processes and demonstrates how agile practices can improve customer satisfaction and value delivery. The study also finds that some agile methods are less collaborative and responsive to changes than others. While Raza's focus lies in comparing agile frameworks, our work emphasizes change management as a whole - examining how change is accepted, communicated, and integrated across project phases and team workflows.

Saher [11] proposes a taxonomy of requirements changes in agile environments, categorizing them by origin and underlying cause. In alignment with this approach, our study places greater emphasis on understanding how these changes are perceived and managed throughout the software development lifecycle. Saher's categorization model supports our effort to systematically identify and analyze different types of changes in agile projects.

Although these studies offer valuable insights into the impacts of requirements changes on various projects dimensions, there is a recurring trend in the literature: most focus primarily on functional requirements, often overlooking other critical changes types, such as technological shifts and non-functional requirements. This limited scope may restrict the comprehensive understanding of change dynamics in software projects. In contrast, the present study expands this view by considering changes related to functional and non-functional requirements, technology, and team structure - providing a more holistic perspective aligned with organization practice.

Chapter 3

Literature Review

A systematic review has three phases i.e. **Planning**, **Conducting**, and **Reporting** [38]. Each of them is detailed in the following subsections.

3.1 Planning Phase

In the planning phase, research questions were developed to guide the study's development. The objective of this review protocol is to identify how small and medium-sized organizations handle change management and the impacts of these changes on agile software development projects. It seeks to understand the main changes the project is likely to face, as well as their impacts and outcomes. Additionally, the study aims to understand how communication between the team and stakeholders occurs in relation to these changes. To gather the necessary information, the Research Questions (RQs) outlined in 3.1 were addressed.

Table 3.1: Research Questions

ID	Description
RQ1	What types of changes have the greatest impact on software development projects?
RQ2	What are the impacts of changes during software development projects?
RQ3	Does the use of agile methodologies help manage, accept, and implement these changes in a way that does not significantly impact the project?
RQ4	How are changes managed and communicated to the team?

Based on the research questions, the scope of the review was defined using the PICO approach, as in 3.2. PICO is an acronym for ***Population***, ***Intervention***, ***Comparison***, and ***Outcomes***. These four components are fundamental for formulating the research question and constructing a bibliography search strategy [33].

Table 3.2: PICO Framework

Component	Description
Population	Agile method
Intervention	Change management
Comparison	*Not applicable*
Outcome	Impact

To define the review search string, the following keywords and their synonyms were considered according to the presented PICO structure 3.3.

Table 3.3: Search Terms and Synonyms

Keyword	**Synonyms**	**Related to**
Agile method	Agile development | Agile method | Agile software development	Population
Change management	Change control	Intervention
Impact	Impacts	Outcome

Considering the terms in Tables 3.2 and 3.3, the search string was structured using the Boolean operator "OR" for the synonymous and alternative terms presented in Table 3.3, concatenated with the PICO category using the operator "AND". The resulting search string was:

("Agile method" OR "Agile development" OR "Agile software development") AND ("Change management" OR "Change control") AND ("Impact" OR "Impacts")

To gain an initial understanding of how agile methodologies adopt and accept changes and how change management is applied within software organizations, the following control articles were defined:

- Keaveney, S.; Conboy, K. (2006). Cost estimation in agile development projects. Proceedings of the 14th European Conference on Information Systems, ECIS 2006 [21].

- Raza, S.; Waheed, U. Managing change in agile software development a comparative study. In: 2018 IEEE 21st International Multi-Topic Conference (INMIC) [37].[S.l.:s.n.], 2018. p. 1–8.

- Saher, N.; Baharom, F.; Ghazali, O. Requirement change taxonomy and categorization in agile software development. In: . [s.n.], 2017. v. 2017-November, p. 1 – 6 [11].

The search string was executed in the following electronic databases:

- IEEExplore (http://ieeexplore.ieee.org)
- Scopus (www.scopus.com)

The following inclusion and exclusion criteria 3.4 were defined for the selection of studies. The review includes every paper returned by the search string that meets all the following inclusion criteria (IC) and does not meet any option of the exclusion criteria (EC).

Table 3.4: Inclusion and Exclusion Criteria

	Inclusion Criteria
IC1	Articles discussing the impact of change management on software development.
IC2	Publications presenting challenges or mitigation strategies related to change management.
IC3	Studies addressing the application of agile methodologies in change contexts.
IC4	Studies focused on change management in software development projects.
	Exclusion Criteria
EC1	Articles not addressing agile methodologies or discussing only technical aspects unrelated to change management.
EC2	Articles not including comparisons or analyses of the impacts of changes.
EC3	Publications not discussing challenges or mitigation strategies in change management.
EC4	Studies exclusively focused on large enterprises or industrial contexts not applicable to small and/or medium-sized enterprises.
EC5	Studies not focused on change management in software development projects.
EC6	Articles not written in English.
EC7	Non-peer-reviewed technical reports.
EC8	Publications about some conference proceeding, journal, or some other collection of papers/publications.
EC9	Publications whose full text is not available for download, in their complete form, in digital libraries, or through any other way without cost to the researcher.

The quality assessment questions in 3.5 were defined and answered for each selected publication/article before the extraction process, to classify/score the results considering six questions. The scores were: 5 points for “Yes”; 2.5 for “Partially”; and 0 for “No”. Based on this scoring, each publication could obtain between 0 to 30 points, with a cut-off score for approval of 15 points.

Table 3.5: Quality Assessment Questions

ID	Description	Score
QA1	Does the study directly address change management in agile software development projects?	Yes/No/Partially
QA2	Are the study participants representative of the context of interest (e.g., agile development, small or medium enterprises, team)?	Yes/No/Partially
QA3	Are the study's conclusions relevant and do they have practical implications for software project management?	Yes/No/Partially
QA4	Does the article address how to handle change throughout the software development lifecycle (e.g., requirement changes, technological advancements, or team changes)?	Yes/No/Partially
QA5	Does the study discuss challenges in change management such as resistance to change, communication gaps, or implementation obstacles?	Yes/No/Partially
QA6	Does the article evaluate how changes impact project performance, including quality, adherence to schedule, and/or cost?	Yes/No/Partially

After answering the research question 3.5, each approved publication/article underwent a data extraction process to collect essential information to answer the research questions. The questions in 3.6 were developed for data extraction:

Table 3.6: Data Extraction Questions

Description	Type	Values
DE1 – Study Title	String Field	–
DE2 – Authors	String Field	–
DE3 – Year of Publication	Integer Field	–
DE4 – Source of Publication	String Field	–
DE5 – Type of Study	Select One Field	Case Study / Experiment / Literature Review / Other
DE6 – Main objectives of the study	String Field	–
DE7 – Does the study address change management in agile software development projects?	Select One Field	Yes / No
DE8 – Types of changes discussed	Select Many Field	Requirement Changes / Team Changes / Technological Advancements / Other
DE9 – Does the study discuss challenges in change management?	Select One Field	Yes / No
DE10 – Specific challenges mentioned	Select Many Field	Communication Gaps / Implementation Obstacles / Resistance to Change / Other
DE11 – Does the study assess how change impacts project performance?	Select One Field	Yes / No
DE12 – Aspects of project performance evaluated	Select Many Field	Cost / Quality / Schedule Adherence / Other
DE13 – Does the study provide practical recommendations for change management in agile projects?	Select One Field	Yes / No

3.2 Conducting Phase

After the planning phase, the review protocol was applied, starting on May 29, 2024, with the following steps:

1. **Search String Execution**: The search string was run in electronic databases, and the resulting publications were cataloged for further analysis.

2. **Results Merging**: The results from all databases were combined using a tool for support.

3. **First Filter - Removing Duplicates**: Duplicate results were identified and removed.

4. **Second Filter - Title, Abstract, and Introduction Analysis**: The remaining results were reviewed based on their titles, abstracts, and introductions, following inclusion/exclusion criteria. Irrelevant results were excluded. To minimize the risk of excluding relevant studies too early, both a specialist and a software engineering student assessed whether each result should be included or excluded.

5. **Third Filter - Full Text Analysis**: Papers selected from the previous step were fully reviewed to determine whether they should be included for further analysis.

6. **Data Extraction**: The papers selected from Step 5 were evaluated using the Quality Assessment Form 3.5, papers scoring above 15 points were analyzed, and the Data Extraction Form 3.6 was applied. The results obtained on each step are detailed in the following.

7. **Snowballing**: The Snowballing method was applied using the three previously defined control articles. First, all references cited in these articles were analyzed to identify those containing the main keywords: *Change Management*, *Change Management in Agile Software Development Projects*, and *Impact of Changes in Agile Software Development Projects*. This initial screening resulted in the identification of 15 potentially relevant articles for the review, which proceeded to the next phase of evaluation.

1. **Search String Execution**: In this stage, the search sequence was executed in

the databases, and the results obtained were cataloged using Parsif.al [1] for future analysis. The obtained results are presented in 3.1.

2. **Results Merging**: A total of 170 results were obtained, which were merged/imported into Parsif.al (https://parsif.al).

3. **First Filter - Removing Duplicates**: Using Parsif.al, 7 duplicate results were removed.

4. **Second Filter - Title and Abstract Analysis**: Considering the inclusion and exclusion criteria, the results were analyzed based on their titles, abstracts, and introductions. After applying these filters, 130 results were rejected, and 33 were accepted.

5. **Third Filter - Full Text Analysis**: This step involved a full reading of the accepted results to answer the quality assessment questions. After this stage, of the 33 results accepted, 15 reached the defined cut-off score 3.5, while 18 did not meet the minimum score.

6. **Data Extraction**: The results that reached the cut-off score were evaluated to answer the question presented in 3.6.

7. **Snowballing**: In this stage, the 15 articles previously identified as potentially relevant were analyzed in their entirety. Each article was evaluated based on the established inclusion and exclusion criteria, as well as quality assessment questions. After this analysis, 3 articles were approved and included in the review, while 12 were excluded for not meeting the criteria.

[1] https://doi.org/10.5281/zenodo.15564616

Table 3.7: Search Results

Source	Seach String	Imported Studies
IEEE Digital Library	("Agile method" OR "Agile development" OR "Agile software development") AND ("Change management" OR "Change control") AND ("Impact" OR "Impacts")	23
Scopus	("Agile method" OR "Agile development" OR "Agile software development") AND ("Change management" OR "Change control") AND ("Impact" OR "Impacts")	147
Total		**170**

The table 3.8 below summarizes the total number of articles obtained after each mentioned step:

Table 3.8: Summary of the search results

Stages	IEEE Xplore	Scopus	Total of studies included
Total retrieved	23	147	170
Removing duplicates	0	7	163
Initial selection	7	16	33
Second selection	4	11	15
Snowballing	0	3	3
Total Final of Selected Articles: 18			

3.3 Reporting Phase

Considering that 15 papers were selected as the result of the review execution, this means that 9,20% of the papers initially obtained (after removing the duplicates) contributed to this review. Considering the previously defined research questions (RQ), the following analyses were performed:

3.3.1 RQ1: What types of changes have the greatest impact on software development projects?

Based on the systematic literature review, which included 18 articles selected after applying quality criteria, four main types of changes with the greatest impact on software project development were identified, as detailed in table 3.9.

Among these types, changes related to software product requirements stand out as the most discussed in the literature, receiving more emphasis compared to changes in software project requirements.

All 18 analyzed articles point to requirements (both functional and non-functional) as the factor with the most significant influence on change requests throughout software development. Requirements Engineering (RE) plays a central role in this context, as the success of a project directly depends on the clarity and precision of the defined requirement. Unlike the traditional model, where RE is treated as an initial phase, in agile development, this process is continuous, carried out throughout the entire development cycle.

According to [39], requirements change is defined as "the tendency for requirements to change over time, reacting to the evolving needs of customers, stakeholders, organizations, and the work environment." These changes occur due to factors such as shifts in business domains, technological advancements, or changing customer needs. At the beginning of the project, requirements are often not fully defined, making changes during development predictable. According to [40], changes in the Software Lifecycle (SLC) have a lower cost than changes in requirements that occur in later phases.

Although most research focuses on functional requirements, the article "Identification of Sustainability Characteristics and Sub-Characteristics as Non-Functional Requirements for Requirement Change Management in Agile" [41] addresses the impacts of ignoring Non-Functional Requirements (NFRs) during the requirement change process. When neglected, these requirements can also cause adverse effects on software products, reducing quality and increasing project costs. However, Requirement Change Management in Agile (RCM) does not provide effective methods or approaches for dealing with changes related to non-functional requirements.

The unstable nature of business requirements often increases the development budget and poses a significant risk to the project schedule. In summary, requirement change is considered the main factor contributing to project failure [10].

Table 3.9: Types of changes that most impact software project development identified from the literature.

Types of Changes	**Studies**	**# Papers**
Functional Requirements	[1], [3], [10], [11], [34]–[36], [38], [42]–[46]	13
Non-Functional Requirements	[47]	1
Technology	–	0
Team	–	0

3.3.2 RQ2: What are the impacts of changes during software development projects?

Based on the selected articles, we have the following classification of the identified impact, as present in table 3.10. However, the obtained data is not entirely conclusive regarding the precise measurement of the impacts caused by changes throughout the software development lifecycle. It was not possible to determine, in terms of percentages or degree of importance, how these impacts are felt by team, clients, and/or stakeholders during the various phases of the project.

Table 3.10: Types of impacts according to changes on software development projects from literature

Types of impact	**Studies**	**#papers**
Cost	[3], [34], [35], [38], [42], [44], [45], [47]	8
Schedule	[10], [35], [42], [43], [44], [45]	6
Scope	[45]	1
Rework	[34],	1
Quality	[1], [3], [11], [42], [43], [44], [45], [46], [47],	9
Workflow	[36], [42]	2

Despite this limitation, some relevant conclusions can be drawn:

1. **Agile Methods and Responsiveness to Change**: Development using agile methodologies shows a significantly better ability to respond to changes during the project, compared to traditional development approaches [35] [34].

2. **Focus on Change Control and Scope Increase**: Agile methodologies tend to focus on change control rather than analyzing their impact. This is directly related to scope increase, which is a common consequence of changing requirements [9] [10] [40].

3. **Impact on Cost**: Changing requirements are identified as one of the main causes of cost estimation problems in agile software development projects [37].

4. **Adverse Effects on the Product**: To varying degrees, changes during development can create adverse effects on the product, affecting both quality and schedule [11] [43] [48].

3.3.3 RQ3: Does the use of agile methodologies help to manage, accept, and implement these changes in a way that does not significantly impact the project?

One of the key principles of the Agile Manifesto is to "welcome changing customer needs, even late in development". This reflects the flexibility of agile methodologies, which allow changes to be accepted at any stage of the software, with the primary goal of ensuring customer satisfaction [11]. In contrast, traditional methodologies tend to resist changes due to rigid planning and extensive documentation.

Of the 18 articles selected and approved based on quality criteria, all discuss change management in the context of software development using agile methodologies. Some of these studies analyze how changes should be received and implemented, taking into account the priority and scale of change requests. They emphasize that agile development, with its focus on user involvement and project feasibility, is an effective approach for managing changes in small to medium-scale projects. This, agile methods are seen as a solution to address evolving organizational needs and increase customer satisfaction.

Agile methodologies, such as SCRUM, are increasingly used in software development as a response to the challenges of managing frequent changes. However, it is important to note that Agile-based projects can produce unsatisfactory results when there is a lack of a well-defined change evaluation process [17]. As discussed in “An Empirical Investigation of Factors Causing Scope Creep in Agile Global Software Development Context: A Conceptual Model for Project Managers” [9], the easy acceptance of changes in the agile process can compromise quality, increase costs, and alter planning. This can negatively impact project success, as there is a greater focus on controlling and accepting scope changes rather than analyzing their impact on cost and quality.

On the other hand, by focusing on the prioritization of major change requests, the project team can optimize the use of its resources, improve software quality, and reduce the time needed to bring the product to market [10]. This approach aligns with the principles of agile development, which seek to improve customer satisfaction, reduce delivery time,

and increase business value.

3.3.4 RQ4: How are changes managed and communicated to the team?

According to [49], communication during changes must occur at multiple levels. Not only do business teams need to communicate with each other, but also within their own teams and with other groups that might influence the project.

Strong and close communication is essential for completing projects on time and within budget [35]. However, in the context of Global Software Development, communication becomes more challenging due to language barriers and cultural differences.

In agile methodologies, internal communication within teams is frequent and intense. However, not all changes are reported to clients and/or stakeholders [48]. Technological changes, for example, are generally communicated to clients only if they were previously aware of the chosen technology. Team changes are communicated to clients only when they directly affect the relationship or project delivery.

3.3.5 Results from the Systematic Literature Review (SLR)

Regarding RQ_1, three main types of changes were identified as having the greatest impact on software development projects. Table 3.11 summarize the change types reported in the selected studies. Among the 18 papers analyzed:

- **72% (13 out of 18)** identified **functional requirements** as the most impactful type of change;
- Only **5% (1 out of 18)** cited **non-functional requirements**;
- **23% (4 out of 18)** ([9], [50], [51], [52]) did not specify any type of change.

Table 3.11: Types of changes that most impact software project development identified from literature.

Types of Changes	**Studies**	**# Papers**
Functional Requirements	[1], [3], [10], [11], [34]–[36], [38], [42]–[46]	13
Non-Functional Requirements	[47]	1
Technology	–	0
Team	–	0

For RQ_2, the types of impacts associated with these changes are summarized in the Table 3.12. The six most frequently mentioned impact categories were:

- **Quality**: cited by 50% (9 out of 18) of the studies;
- **Cost**: 44% (8 out of 18).
- **Schedule**: 33% (6 out of 18);
- **Workflow**: 11% (2 out of 18);
- **Scope** and **Rework**: each mentioned in only one study.
- As with change types, **23% (4 out of 18)** of studies ([9], [50], [51], [52]) did not mention specific impacts.

Table 3.12: Types of impacts according to changes on software development projects from literature

Types of impact	**Studies**	**#papers**
Cost	[3], [21], [34], [35], [42], [44], [45], [47]	8
Schedule	[10], [35], [42], [43], [44], [45]	6
Scope	[45]	1
Rework	[34],	1
Quality	[1], [3], [11], [42], [43], [44], [45], [46], [47]	9
Workflow	[36], [42]	2

The literature also reveals a notable gap regarding **how project changes are communicate among teams and stakeholders**. While some studies address decision-making roles (e.g., who approves or rejects changes), the communication mechanisms - such as tools used, frequency, and channels - remain largely underexplored. This observation motivated the development of the survey to further investigate these aspects in practice.

Chapter 4

Survey

Following the Systematic Literature Review (SLR), an empirical survey was conducted with the aim of deepening the understanding of how changes are experienced in software development projects. Specifically, the survey sought to:

- Identify the most frequent types of change requests in software projects;
- Assess the perceived impact of these changes on the project process and overall quality.
- Understand how such changes are managed and communicated throughout the software development lifecycle.

The survey instrument consisted of 40 questions, organized into the following five sections (see Appendix D for the full survey) :

1. **Instructions and Informative Term**: Presentation of the research context, its objectives, and information about the handling of participants' responses.
2. **Respondent Profile**: Collection of general information about participants' academic and professional experience, focusing on project and change management, as well as agile methodologies used.

3. **Change Management Practices**: Identification of practices in managing changes in software projects.

4. **Impacts and Recommendations**: Investigation of the effects of changes on the development cycle and suggestions for improving their management.

5. **Space for Additional Recommendations**: Open field for observations and suggestions to enhance change management.

To guide the design of this empirical investigation, the Goal-Question-Metric (GQM) model [53]. The goal of the study can be formally stated as follows: *"Analyse **requirements change** for the purpose of **evaluation** with respect to **types, impacts, communication** from the point of view of **professionals** in the context of **real projects of software development**."*

In alignment with this goal, the following Guiding Questions (GQs) were defined:

- GQ_1**:** What is the professionals' profile?

- GQ_2**:** What are professionals' perceptions of the types of changes that have the greatest impact on software development projects?

- GQ_3**:** What are professionals' perceptions on the impacts of changes on software development projects?

- GQ_4**:** What are professionals' perceptions of using agile methodologies to help manage changes in software development projects?

- GQ_5**:** What are professionals' perceptions of how requirements changes are managed and communicated?

4.0.1 Survey Results and Analysis

This section presents and analyzes the main findings of the empirical survey conducted with professional involved in software development projects. Table 4.1 summarizes key

characteristics of the respondents, including Academic Training (ACT), Organization Size (Org. Size), Experience Project Management (EPM), Experience Change Management (ECM) and Agile Methodology (AM).

Respondent Profile and Context

A total of **32 professionals** participated in the survey. After validation, **with 31 responses** were considered for analysis. Participants held various position in their organizations, including:

The majority of respondents (30) were based in Brazil, with one respondent from Portugal.

4.0.2 RQ1: What types of changes have the greatest impact on software development projects?

For this RQ, we present the professionals' perception of the changes that have most significant impact on software development projects. The Table 4.2 and 4.3 presents the most frequent changes and changes most likely to compromise the schedule, budget, or quality, respectively.

By comparison these findings with the SLR, it is possible to reaffirm that changes in software product requirements are the most critical. According to [3], requirement changes arise due to evolving business needs, technological advancements, or shifts in customer priorities. Since initial requirements are often incomplete, changes become inevitable throughout the Software Lifecycle (SLC). As highlighted by [12], addressing requirement changes as early as possible in the software lifecycle significantly reduces costs compared to late-stage modifications. The six main impacts identified by professional regarding these changes are: (i) Additional time required; (ii) High rework costs; (iii) Impact on business rules; (iv) Impact on project scope; (v) Impact on the schedule; and (vi) Knowledge gap with the team.

Table 4.2: Distribution of most frequent changes identified from survey per professionals on their projects: FR (Functional Requirements), NFR (Non-Functional Requirement).

Professionals	FR	NFR	Technology	Team	Scope
P1 - Dev					
P2 - Dev					
P3 - Dev					
P4 - Dev					
P5 - Dev					
P6 - Dev					
P7 - Dev					
P8 - Dev					
P9 - Dev					
P10 - Dev					
P11 - Dev					
P12 - Dev					
P13 - Dev					
P14 - Dev					
P15 - Dev					
P16 - Director of Dev					
P17 - Director of Technology					
P18 - PO					
P19 - PO					
P20 - PO					
P21 - PM					
P22 - PM					
P23 - PM					
P24 - PM					
P25 - QA					
P26 - RA					
P27 - RA					
P28 - SM					
P29 - Researcher					
P30 - Researcher					
P31 - TL					

Table 4.3: Distribution of changes that most impact software project development from survey per professionals on their projects.

Profissionals	FR	NFR	Technology	Team
P1 - Dev				
P2 - Dev				
P3 - Dev				
P4 - Dev				
P5 - Dev				
P6 - Dev				
P7 - Dev				
P8 - Dev				
P9 - Dev				
P10 - Dev				
P11 - Dev				
P12 - Dev				
P13 - Dev				
P14 - Dev				
P15 - Dev				
P16 - Director of Dev				
P17 - Director of Technology				
P18 - PO				
P19 - PO				
P20 - PO				
P21 - PM				
P22 - PM				
P23 - PM				
P24 - PM				
P25 - QA				
P26 - RA				
P27 - RA				
P28 - SM				
P29 - Researcher				
P30 - Researcher				
P31 - TL				

4.0.3 RQ2: What are the impacts of changes during software development projects?

Although the Systematic Literature Review (SLR) provided a theoretical classification of impacts, the survey results offer empirical evidence that reinforces and expands upon

these findings. Table 4.4 presents the distribution of impacts based on the perceptions of the surveyed professionals.

From the survey responses, it is evident that changes most frequently affect the following project dimensions:

- **Schedule**: Cited by 26 participants (83%).
- **Rework**: Cited by 19 participants (61%).
- **Cost**: Cited by 14 participants (45%).
- **Quality**: Cited by 16 participants (51
- **Scope**: Cited by 13 participants (41%).
- **Workflow**: Cited by 11 participants (35%).

These results confirm that **schedule**, **rework**, and **cost** are the areas most vulnerable to change impacts in software development projects.

Regarding the frequency of impacts:

- **Frequently or very frequently**: 15 participants (48%) reported that changes impact projects frequently or very frequently.
- **Occasionally**: 10 participants (32%).
- **Rarely or never**: 6 participants (20%).

Concerning the degree of impact on project quality:

- **Improvements** were reported by 16 participants (51%), with 4 (13%) indicating significant improvements.
- **No significant interference** was indicated by 9 participants (29%).
- **Reductions in quality** were observed by 6 participants (19%).

Additionally, the survey revealed that:

- **Changes impacting schedule** are the most critical concern, being reported by nearly all roles (developers, PMs, POs, and researchers).

- **Cost** is consistently liked to schedule overruns, corroborating theoretical models such as the Standish Group's Chaos Report findings.

- **Rework** emerge as significant impact are, confirming observations from [34] about hidden rework costs in agile projects.

Finally, concerning change approval and resistance:

- The data shows that **high-impact changes are generally accepted** without significant resistance.

- Decision.making about changes primarily rests with clients and stakeholders, rather than solely with the internal project team, suggesting that **external pressures and client-drive adaptions** are dominant factors in the acceptance process.

In general, the survey demonstrates that although change introduces challenges-especially regarding schedule, rework, and cost-organizations often experience **positive or neutral effects on software quality**, provided that changes are well communicated and systematically managed.

Table 4.4: Distribution of how changes affect projects based on survey responses from professionals.

Professionals	**Impacts**	Frequency of Impact	Degree of Impact
P1 - Dev	Schedule; Cost; Quality	Frequently	Reduces
P2 - Dev	Schedule	Rarely	Does not Interfere
P3 - Dev	Schedule; Cost; Scope; Rework	Rarely	Does not Interfere
P4 - Dev	Cost; Workflow; Quality; Rework; Schedule	Never	Does not Interfere
P5 - Dev	Scope; Rework; Schedule; Cost	Occasionally	Does not Interfere
P6 - Dev	Workflow; Quality; Rework	Frequently	Reduces
P7 - Dev	Rework; Quality; Schedule	Rarely	Improves
P8 - Dev	Rework	Occasionally	Improves
P9 - Dev	Schedule	Occasionally	Improves
P10 - Dev	Schedule	Occasionally	Improves
P11 - Dev	Scope; Rework; Quality	Occasionally	Improves
P12 - Dev	Workflow; Quality; Rework	Frequently	Improves
P13 - Dev	Schedule; Cost; Quality; Rework	Very Frequent	Improves Significantly
P14 - Dev	Schedule; Cost; Workflow; Quality; Rework	Occasionally	Improves
P15 - Dev	Schedule	Frequently	Improves
P16 - Director of Dev	Schedule; Cost; Workflow; Scope; Quality; Rework	Very Frequent	Improves Significantly
P17 - Director of Technology	Scope; Schedule; Quality; Rework	Frequently	Improves Significantly
P18 - PO	Schedule; Scope	Occasionally	Reduces
P19 - PO	Worflow	Never	Improves Significantly
P20 - PO	Workflow; Schedule; Cost	Rarely	Does not Interfere
P21 - PM	Schedule; Cost; Scope	Frequently	Does not Interfere
P22 - PM	Schedule; Cost; Scope; Quality; Rework	Occasionally	Reduces
P23 - PM	Schedule; Cost; Rework	Rarely	Does not Interfere
P24 - PM	Schedule; Scope; Quality; Rework	Frequently	Improves
P25 - QA	Rework	Rarely	Improves
P26 - RA	Schedule; Cost; Scope; Workflow	Rarely	Does not Interfere
P27 - RA	Schedule; Workflow; Quality; Rework	Frequently	Reduces
P28 - Researcher	Schedule; Cost; Quality; Rework	Occasionally	Does not Interfe
P29 - Researcher	Schedule; Scope; Rework	Rarely	Does not Interfere
P30 - SM	Schedule; Scope	Occasionally	Reduces
P31 - TL	Schedule; Rework; Workflow; Scope	Occasionally	Reduces

4.0.4 RQ3: Does the use of agile methodologies help manage, accept, and implement these changes in a way that does not significantly impact the project?

One of the fundamental principles of the Agile Manifesto is to ***"Welcome changing customer needs, even late in development"*** [2]. This principle emphasizes the flexibility inherent in agile methodologies, which advocate for the continuous acceptance and integration of changes at any stage of the project lifecycle to better customer needs.

The Systematic Literature Review (SLR) reinforced this concept by identifying that several studies analyze how changes should be evaluated, prioritized, and incorporated based on their criticality and potential impact on the project. Studies such as [20], [11] suggest that agile practices, when properly implemented, facilitate more adaptive and resilient project environments capable of absorbing change without significant disruptions.

However, it is crucial to recognize that the adoption of agile methodologies does not automatically ensure effective change management. As discussed in [9], the indiscriminate acceptance of changes can lead to unintended consequences, including compromised product quality, increased cost, and disrupted planning. Therefore, while agility supports flexibility, it also requires disciplined processes for evaluating and managing change requests.

The survey findings corroborate this nuanced perspective. Most respondents indicated the use of agile methodologies, predominantly Scrum, often in combination with other frameworks such as Kanban and Lean. Tools such as as Jira and Trello were commonly cited as critical enablers for managing and organizing changes within the backlog, allowing teams to maintain structure, prioritize effectively, and monitor the impact modifications.

Despite the structured processes available, the survey revealed some challenges:

- High-impact changes rarely faced resistance within the surveyed organizations.
- Many changes were directly incorporated into the project backlog, often without a formalized impact analysis process.

- In several cases, changes were initially treated as independent request before undergoing prioritization and effort assessment.

These observations highlight that, although agile methodologies promote a framework conducive to change management, the **maturity of processes** and the **discipline in evaluating changes** remain decisive factors in mitigating negative impacts on cost, schedule, and quality.

In summary, agile methodologies - when combined with structured change evaluation and prioritization practices - can significantly enhance an organization's ability to manage, accept, and implement changes without compromising project success.

4.0.5 RQ4: How are changes managed and communicated to the team?

Effective communication is critical for successful change management, ensuring proper alignment between teams, stakeholders, and clients [4], [11]. The survey results provide a detailed view of how changes are communicated , tracked, and approved in software development projects.

Regarding the **communication of changes**:

- **58% (18 out of 31)** of respondents stated that internat changes are communicated **through formal meetings**.
- **42% (13 out of 31)** reported that **only changes with significant impact** are communicated to the team.

These findings are consistent with the literature, which suggests that while agile environments promote openness to changes, not all types of modifications, particularly technical or minor ones, are systematically communicated to all stakeholders [48].

As for **change tracking mechanisms**:

- **84%** of respondents use **project management tools** (such as Jira, Trello, or internal systems) to document and manage changes.

- A smaller portion, **10%**, still relies on **manual documentation**.

These results highlight a strong reliance on digital tools for maintaining traceability, prioritizing change requests, and improving project transparency. Regarding the **approval processes** for changes:

- **65%** of respondents indicated that changes are subjected to **discussions with impact estimation** prior to decision-making.
- **29%** reported **direct decision-making** by the Product Owner (PO) or Project Manager (PM) without formal discussion.
- Only **3%** accepted changes without any structured evaluation.

These findings suggest that while agile teams embrace changes, there is an effort in many organizations to ensure at least a basic impact assessment before incorporation into the backlog.

In terms of **decision-making authority for changes**:

- The **Project Manager** was cited as the primary decision-maker by **43%** of the participants.
- The **Product Owner** was cited by **23%** of respondents.
- **Clients or external stakeholders** were responsible for change approval in **30%** of the cases.
- A minor percentage (**6%**) indicated that changes requested by the CEO or business area are automatically accepted, regardless of internal analysis.

There results reinforce the idea that in many agile agile environments, although the team plays a significant role, change decisions often remain strongly influenced by external stakeholders or higher management.

In summary, the survey results reveal that:

- Communication of changes is mostly formalized through meetings but tends to prioritize high-impact modifications.
- Project management tools dominate change tracking practices, reducing reliance on manual documentation.
- A structured evaluation of impact before accepting changes is common, although direct decision-making without detailed assessment still occurs in almost a third of cases.
- Change approval authority is generally centered on the Project Manager, Product Owner, or stakeholders.

Table 4.1: Organized by position: Developer (Dev), Director of Dev, Director of Technology, Product Owner (PO), Product Manager (PM), Quality Assurance (QA), Requirements Analytic (RA), Scrum Master (SM), Teach Leader (TL), and Researcher. Regarding ACT, the categories are: Incomplete Higher Education (IHE), Higher Education (HE), Postgraduate/ MBA, Master's Degree, and Doctorate.

Prof.	Position	Seniority	ACT	Org. Size	EPM	ECM	AM
P1	Dev	Mid-Level	HE	Medium	1-3 years	Less than 1 year	Kanban
P2	Dev	Senior	IHE	Medium	1-3 years	NOE	Kanban
P3	Dev	Senior	IHE	Medium	5-10 years	NOE	Scrum
P4	Dev	Junior	IHE	Medium	Less than 1 year	Less than 1 year	Kanban
P5	Dev	Junior	IHE	Medium	Less than 1 year	1-3 years	Scrum
P6	Dev	Junior	HE	Medium	Less than 1 year	NOE	Scrum
P7	Dev	Mid-Level	HE	Medium	1-3 years	1-3 years	Scrum
P8	Dev	Mid-Level	IHE	Medium	1-3 years	1-3 years	Scrum
P9	Dev	Mid-Level	HE	Medium	Less than 1 year	NOE	Scrum and Kanban
P10	Dev	Mid-Level	HE	Medium	Less than 1 year	NOE	Scrum, Kanban and Lean
P11	Dev	Junior	IHE	Small	1-3 years	NOE	
P12	Dev	Junior	IHE	Small	1-3 years	NOE	
P13	Dev	Junior	Master's Degree	Medium	1-3 years	NOE	Kanban
P14	Dev	Junior	HE	Small	Less than 1 year	NOE	Agile and Scrum
P15	Dev	Junior	HE	Medium	1-3 years	1-3 years	Scrum
P16	Director of Dev	Executive/ C-Level	Master's Degree	Medium	+10 years	+10 years	Scrum
P17	Director of Technology	Executive/ C-Level	Postgraduate/ MBA	Medium	5-10 years	5-10 years	Scrum
P18	PO	Speacialist/ Leader	Postgradute/ MBA	Medium	5-10 years	5-10 years	Scrum
P19	PO	Mid-Level	HE	Medium	3-5 years	1-3 years	Scrum
P20	PO	Mid-Level	IHE	Medium	1-3 years	1-3 years	Kanban
P21	PM	Senior	Postgraduate/ MBA	Medium	+10 years	+10 years	Scrum and Kanban
P22	PM	Speacialist/ Leader	HE	Medium	+10 years	+10 years	Scrum and Kanban
P23	PM	Mid-Level	IHE	Small	1-3 years	NOE	Scrum
P24	PM	Senior	HE	Medium	3-5 years	3-5 years	Scrum
P25	QA	Mid-Level	HE	Small	NOE	NOE	Scrum
P26	RA	Senior	Postgraduate/ MBA	Medium	1-3 years	1-3 years	Scrum
P27	RA	Junior	IHE	Medium	1-3 years	1-3 years	Scrum
P28	SM	Specialist/ Leader	Postgradute/ MBA	Medium	5-10 years	5-10 years	Figh Level
P29	Researcher	Specialist/ Leader	Doctorate		3-5 years	5-10 years	Scrum
P30	Researcher	Senior	Doctorate		3-5 years	3-5 years	Scrum and Lean
P31	TL	Mid-Level	IHE	Small	3-5 years	1-3 years	Scrum and Kanban

Chapter 5

Proposed Approach

5.1 Introduction

This chapter presents a guide for managing change in agile software development projects, structured around three checklists developed to support teams in evaluating and controlling change-related aspects throughout the project lifecycle. These checklists were designed to provide clarity, traceability, and decision-making support when facing changes in scope, requirements, or general requests for modification.

The proposed guide comprises the following instruments:

- A **Scope Definition Checklist**, which helps verify whether the software project's scope is well-defined, feasible, and aligned with business goals;
- A **Requirements Gathering Checklist**, which assists in validating the clarity, testability, traceability, and relevance of functional and non-functional requirements;
- A **Change Management Checklist**, which supports the analysis, communication, and decision-making process associated with proposed changes during development.

Each checklist includes questions that must be answered collaboratively by relevant project stakeholders. These questions aim to prevent ambiguity, promote alignment between business and technical objectives, and mitigate the risks of rework, scope creep,

and poorly communicated changes.

To ensure the practical relevance of the checklists, their design was grounded in two key sources:

- The results of the Systematic Literature Review (SLR) presented in Chapter 3, which identified the most common types of changes and best practices for handling them;
- The empirical findings from the survey (Chapter 4), which captured how professionals across different roles and organizations perceive, communicate, and manage in agile environments.

In this chapter, we also present a traceability matrix, linking each checklist question to its origin in the literature or survey findings, reinforcing the empirical and theoretical foundation of the proposed tools. Additionally, we analyze how these instruments align with agile roles and responsibilities, particularly in Scrum, and demonstrate how the checklists can be integrated into the agile workflow to support structured and collaborative change management.

5.2 Approach Artifacts

This section presents the practical instruments developed as part of this study to support change management in agile software projects. These artifacts consist of three checklists and two change request templates:

- **Scope Definition Checklist** A: support validation and clarity of project scope.
- **Requirements Gathering Checklist** B: ensure quality and traceability of functional and non-functional requirements.
- **Change Management Checklist** C: guides the evaluation and handling of change requests during development.

- **Change Request Template - Complete Version C**: detailed form to record all relevant information for impact and decision analysis.
- **Change Request Template - Simplified Version C**: minimal and agile-friendly version for fast tracking and team collaboration.

To ensure clarity and document readability, the full content of these artifacts is included in A to C. Each checklist was structured based on theoretical and empirical evidence, as demonstrated in Section 5.2.1, and is designed to be integrated into the agile development process, as explained in Sections 5.3.2 and 5.3.3.

5.2.1 Checklists Foundations and Traceability

To ensure that each question in the proposed checklist is grounded in theoretical and empirical evidence, a traceability matrix was developed. This matrix relates each checklist item to its respective source in the Systematic Literature Review (SLR) and, where applicable, to the corresponding question(s) used in the survey conducted in this study.

The traceability matrix is organized by checklist type: Scope Definition, Requirements Gathering, and Change Management. Each table presents:

- The checklist item (question);
- The corresponding source(s) from the SLR;
- The ID(s) of related survey question(s), when applicable.

This approach reinforce the methodological rigor of the proposed guide and highlights its connection to both academic literature and practical insights from professionals.

Traceability Matrix - Scope Definition Checklist

This traceability matrix 5.1 presents the relationship between each item in the Scope Definition Checklist A, its supporting references from the Systematic Literature Review (SLR), and the corresponding question(s) from the survey when applicable.

Table 5.1: Traceability Matrix - Scope Definition Checklist

Checklist	Checklist Question	Source (SLR)	Survey Question ID
Scope Definition	Is the main deliverable clearly defined, including objectives and core functionalities?	[9], [11], [54]	-
Scope Definition	Are out-of-scope elements identified to avoid ambiguity?	[9], [11], [52]	-
Scope Definition	Have the stakeholders and end-users been identified, and are their needs considered?	[9], [34], [35], [48], [50], [52], [54], [55]	-
Scope Definition	Are the expected benefits (economic, operational, or strategic) clearly stated?	[9], [11], [35]	-
Scope Definition	Have all technical, legal, operational, financial, and timeline constraints been identified?	[3], [9], [11], [35], [48]	-
Scope Definition	Are system integrations and required interfaces specified or planned?	-	-
Scope Definition	Does the scope include performance, reliability, or other critical quality requirements?	[35]	-
Scope Definition	Has the scope's technical, operational, and economic feasibility been evaluated by the team?	[9], [11]	-
Scope Definition	Is the scope aligned with business goals and validated by stakeholders?	[9], [11], [35]	-
Scope Definition	Are the measurable criteria to define project success and acceptance?	[34], [35]	-

Traceability Matrix - Requirements Gathering Checklist

This traceability matrix 5.2 presents the relationship between each item in the Requirements Gathering Checklist B, its supporting reference form the Systematic Literature Review (SLR), and the corresponding question(s) from the survey when applicable.

Table 5.2: Traceability Matrix - Requirements Gathering Checklist

Checklist	Checklist Question	Source (SLR)	Survey Question ID
Requirements Gathering	Is the requirement written clearly and unambiguous?	[3], [11], [34], [52], [55]	18
Requirements Gathering	Is the requirement understandable by all stakeholders and the technical team?	[11], [43]	18, 19
Requirements Gathering	Is the requirement realistic in relation to current technology and within the project scope?	[9], [43]	-
Requirements Gathering	Does the requirement avoid anticipating implementation decisions (premature design)?	[43], [52], [55]	-
Requirements Gathering	Is the requirement testable (can it be verified through testing or validation)?	[11], [47]	-
Requirements Gathering	Is the requirement necessary and does it meet the real needs of the stakeholders?	[11], [43], [50], [52], [54]	-
Requirements Gathering	Is the requirement traceable to its business objectives or origin?	[3], [11], [36], [46], [48], [56]	25
Requirements Gathering	Does the requirement have flexibility for changes if necessary?	[10], [43], [47]	29, 30
Requirements Gathering	Is the requirement accepted by all involved stakeholders?	[3], [11], [43], [52], [55]	-
Requirements Gathering	Does the requirement directly contribute to the project or product objectives?	[43], [48], [52], [54], [55]	-

Traceability Matrix - Change Management Checklist

This traceability 5.3 matrix presents the relationship between each item in the Change Management Checklist C, its supporting references from the Systematic Literature Review (SLR), and the corresponding question(s) from the survey when applicable.

Table 5.3: Traceability Matrix - Change Management Checklist

Checklist	Checklist Question	Source (SLR)	Survey Question ID
Change Management	Is the request a real change or a specific customization?	[11], [36], [43]	-
Change Management	Does the change refer to a Functional Requirement (RF) or a Non-Functional Requirement (NFR)?	[3], [10], [11], [21], [36], [37], [43], [46]–[48], [50], [52], [54], [55]	18
Change Management	Does the change impact existing requirements (scope, deliverables, or previous versions)?	[9]–[11], [36], [47], [48], [52], [54]	16
Change Management	Is there a direct impact on the schedule, cost, or team effort?	[9], [10], [21], [34], [36], [37], [42], [46], [48], [52]	16, 17, 19, 20, 21, 26, 34, 35, 36
Change Management	Will the change result in significant rework? If so, has it been assessed?	[9], [34], [35], [37], [42], [50]	21, 36
Change Management	Has technical impact and feasibility analysis been conducted?	[11], [24], [36], [37], [42]	16
Change Management	Was the change discussed with the PO and/or Project Manager before the final decision?	[9], [11], [35], [42], [50]	29, 36
Change Management	Was the change recorded and prioritized in the backlog once accepted?	[3], [36], [37], [43], [46], [55]	-
Change Management	Does the change benefit more than one client or stakeholder, or bring collective value?	[35], [36]	-
Change Management	Was there resistance to change? If so, was it discussed and justified?		27, 28

The traceability matrices demonstrates that the checklist items are not arbitrarily defined but are derived from solid research findings and practical evidence. This ensures that the proposed guide is both methodologically source and applicable to real-world agile software development environments.

5.3 Approach Flow and Roles

In agile software development, especially in frameworks like Scrum, change is not only accepted - it is expected. However, without proper structure and shared accountability,

unmanaged changes can lead to misalignment, rework, and delivery failure. This section presents how the proposed checklist integrates into the agile change flow, identifying the responsibilities of each role and ensuring that decisions are informed, collaborative, and traceable.

5.3.1 Agile Roles Involved in the Change Process

In agile teams, change management is a shared responsibility. While roles vary slightly depending on the framework (e.g., Scrum, XP, Kanban), most include at least one member responsible for evaluating, facilitating, approving, or implementing change. Table 5.4 outlines the core roles and their typical responsibilities regarding change requests.

Table 5.4: Agile Roles and Their Responsibilities in Change Management

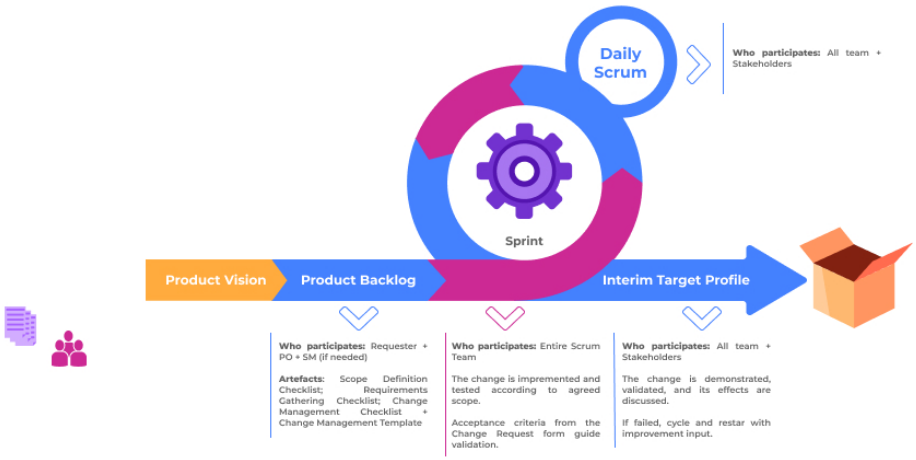
Role	**Responsibilities in Change Management**
Product Owner (PO)	Evaluates the business value of proposed changes, prioritizes them in the backlog, and ensures alignment with stakeholders' goals.
Scrum Master (SM)	Facilitates discussions about changes, removes blockers, and ensures that ceremonies support clear communication of change-related information.
Development Team (Dev)	Assesses the technical feasibility of changes, estimates required effort and potential rework, and implements accepted changes incrementally.
Stakeholders	Propose, approve, or reject changes; provide input and validation throughout the development process.
Quality Assurance (QA)	Validates whether changes are testable, verifies that acceptance criteria are met, and ensures that new changes do not introduce regressions.

5.3.2 Change Flow in Scrum + Checklist Integration

In agile environments, particularly within the Scrum framework, changes are not only expected but embraced as opportunities for improvement. However, without proper structure and accountability, these changes may introduce risks such as misalignment, rework, and delivery delays.

Figure 5.1 illustrates the proposed change flow integrated with agile practices and the use of checklists at each critical step. This visual representation supports understanding of how each phase - from change initiation to sprint review.

Figure 5.1: Change Flow in Scrum Integrated with Checklists



5.3.3 Role Mapping Across Agile Frameworks in the Change Flow

Although Scrum served as the base for the flow presented, other agile frameworks define similar roles with equivalent responsibilities. As illustrated in Figure 5.5, while the nomenclature varies, the core functions related to change evaluation, decision-making, and implementation remain consistent. These roles can be mapped onto the change flow stages, as detailed in Table 5.6.

Table 5.5: Roles Mapping Across Agile Methods

Scrum	**Lean**	**Kanban**	**XP**	**TDD**	**FDD**	**ASD**	**SAFe**
Development Team	Team Member	Team Member	Developer	Developer	Chief Programmer	Developer	Agile Team Member
Product Owner	Value Stream Manager	Service Request Manager / Product Manager	Customer / On-site Customer	Product Owner	Domain Expert / Customer	Customer / Stakeholder	Product Owner / Product Manager
Scrum Master	Lean Coach	Service Delivery Manager	Coach / Tracker	Scrum Master	—	Facilitator / Coach	Scrum Master / Release Train Engineer (RTE)

Table 5.6: Correspondence of roles in different agile methodologies. These roles can be aligned with the change flow present in Table 5.5, ensuring that each stage has responsible agents regardless of the chosen framework.

Change Flow Stage	**Typical Agile Roles Involved**
Change Request Initiation	Product Owner, Service Request Manager, Domain Expert/Customer, Customer, Value Stream Manager
Checklist Application/ Pre-Evaluation	Product Owner, Lean Coach, Agile Team Member, Facilitator, Tracker, RTE
Impact and Feasibility Review	Dev Team, Developer, Chief Programmer, Agile Team, Team Member
Backlog Prioritization and Acceptance	Product Owner, Product Manager, Domain Expert, Customer, Value Stream Manager
Planning and Implementation	Dev Team, Developer, Agile Team Member, Team Member, Chief Programmer
Validation and Review	QA, Product Owner, Stakeholder, Customer/On-site Customer, Domain Expert, Scrum Master

Regardless of the agile methodology adopted, the proposed checklists and change flow can be adapted to the role taxonomy defined by each framework. The essential principle is that each stage of change assessment involves clear accountability, even if roles differ in name or scope.

5.4 Approach Evaluation

To ensure the clarity, relevance, and applicability of the checklists proposed in this study, a content validation process was carried out based on the method by Lynn (1986) [56]. This process involved a panel of five professionals with experience in software development and project management, who evaluated each question from the three checklists: Scope Definition, Requirements Gathering, and Change Management.

Each question was assessed based on the following three criteria:

- **Clarity**: Is the item written in an understandable way?
- **Relevance**: Is the item import for the purpose of the checklist?
- **Adequacy**: Is the item properly formulated for the proposed purpose?

For each criterion, the evaluators used a four-point Likert scale:

1. Strong Disagree
2. Partially Disagree
3. Partially Agree
4. Strongly Agree

According to Lynn's method, the Content Validity Index (CVI or IVC) was calculated for each item as the proportion of experts who rated it 3 or 4 (i.e., "Partially Agree" or "Strongly Agree"). The formula is present in 5.1. A value of $CVI_{item} \geq 0.78$ is considered acceptable. If below this threshold, the item should be reviewed or eliminated. Additionally, the global CVI for each checklist was computed as the average of the item-level CVIs.

$$\text{IVC}_{\text{item}} = \frac{\text{Number of evaluators who rated 3 or 4}}{\text{Total number of evaluators}} \tag{5.1}$$

The complete tables with evaluator responses and calculations for each CVI dimension (Clarity, Relevance, Adequacy), including question-by-question breakdowns, are presented in Appendix E. The complete version of the applied survey form used to build the traceability matrix is also included in Appendix D.

Additionally, an open-ended feedback field allowed experts to suggest improvements or highlight potential problems in formulation, coverage, or terminology.

5.4.1 Expert Panel Profile

The validation panel 5.7 was composed of five professionals selected based on their expertise in areas such as software engineering, project management, and academia. To characterize the panel, the evaluation form collected academic background, field of professional activity, and years of experience in the area.

The profiles are summarized as follows:

Table 5.7: Expert Profile Summary

Prof.	**Academic Background**	**Areas of Expertise**	**Professional Experience**
P1	Information Management	Product Management	8 years
P2	Systems Analysis and Development	Development of web systems, implementation, and technical support in ERP systems	30 years
P3	Systems Analysis and Development	Information Technology / Computer Science, Project Management	+20 years
P4	Master's Degree in Education	Teaching with a focus on software product development	22 years
P5	PhD in Programming Technologies	Computer Science	+30 years

This composition ensured a diverse and qualified evaluation panel, capable of critically assessing the clarity, relevance, and adequacy of the proposed checklist items.

5.4.2 Validation Results - Scope Definition Checklist

The Scope Definition Checklist was evaluated by five professionals with solid experience in software development and project management. Each of the ten items was assessed according to three dimensions - Clarity, Relevance, and Adequacy - using the 4-point Likert scale described earlier.

As shown in Figure 5.2, the Content Validity Index (CVI) was calculated for each item and dimension. Following the guidelines proposed by Lynn (1986) [56], an item is considered valid when it reaches a CVI of 0.78 or higher in all three dimensions.

Among the ten items, seven achieved the maximum score (CVI = 1.00) in all criteria, indicating unanimous agreement among experts regarding the clarity, relevance, and adequacy. The remaining three items also exceeded the minimum threshold (CVI $\geq$ 0.80), except item Q6, which reached exactly 0.80 in all criteria - still within acceptable limits, but signaling a potential need for refinement.

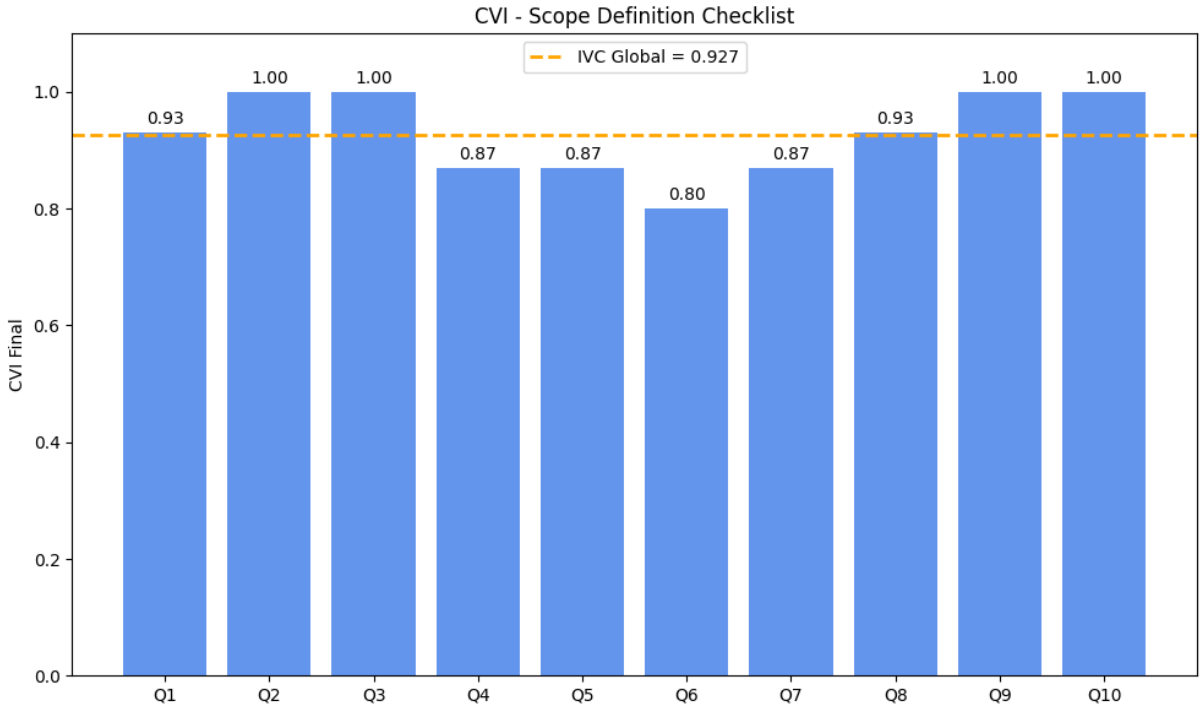
The final CVI for each item was calculated as the average of its Clarity, Relevance, and Adequacy indices. The global CVI of the checklist was calculated by averaging the final CVIs of all ten items, resulting in a value of **0.927**. This score reflects a high degree of content validity for the instrument as a whole.

These results support the conclusion that the Scope Definition Checklist is composed of items that are well formulated, conceptually aligned with the goals of scope definition in agile software projects, and appropriate for practical use in real-world scenarios. Minor improvements may enhance clarity or specificity in certain items, as discussed in the qualitative feedback section.

5.4.3 Validation Results - Requirements Gathering Checklist

The Requirements Gathering Checklist was evaluated by five experienced professionals from the software development field. Each of the ten items was assessed based on the criteria Clarity, Relevance, and Adequacy, using the same 4-point Likert scale and the CVI methodology previously described.

Figure 5.2: CVI Final by Question - Scope Definition Checklist

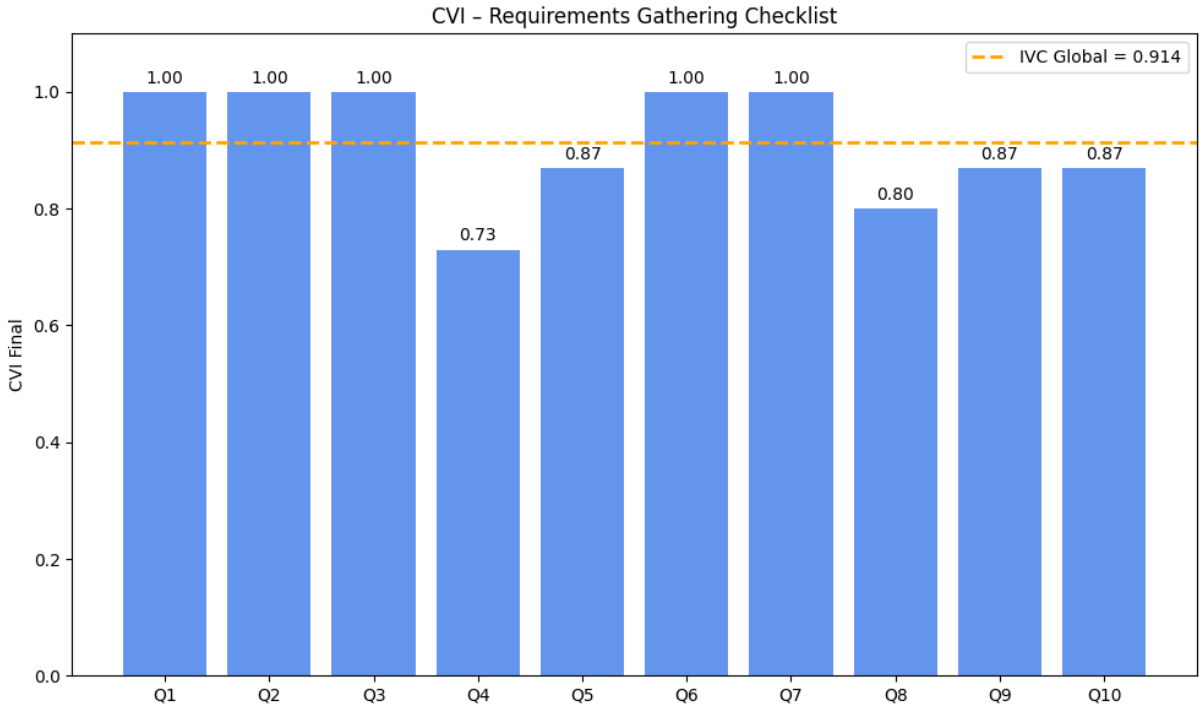


As presented in Figure 5.3, nine out of ten items achieved a CVI equal to or greater than 0.78 in all three dimensions, indicating a high degree of content validity in terms of formulation and alignment with the checklist's intended purpose.

However, item Q4 received a CVI of 0.60 in the Clarity dimension, resulting in a final CVI of 0.73. According to the criteria established by Lynn (1986) [56], this score falls below the recommended threshold of 0.78. Consequently, this item should be reviewed and reformulated, particularly to enhance its clarity for respondents.

Despite this isolated case, the overall CVI for the Requirements Gathering Checklist was calculated as **0.914**, reflecting a strong level of content validity for the instrument as a whole. This result confirms that the checklist is appropriate for professional use and that, with minor adjustments, it can serve as a robust tool to guide and support requirements elicitation processes in agile software development environments.

Figure 5.3: CVI Final by Question - Requirements Gathering Checklist



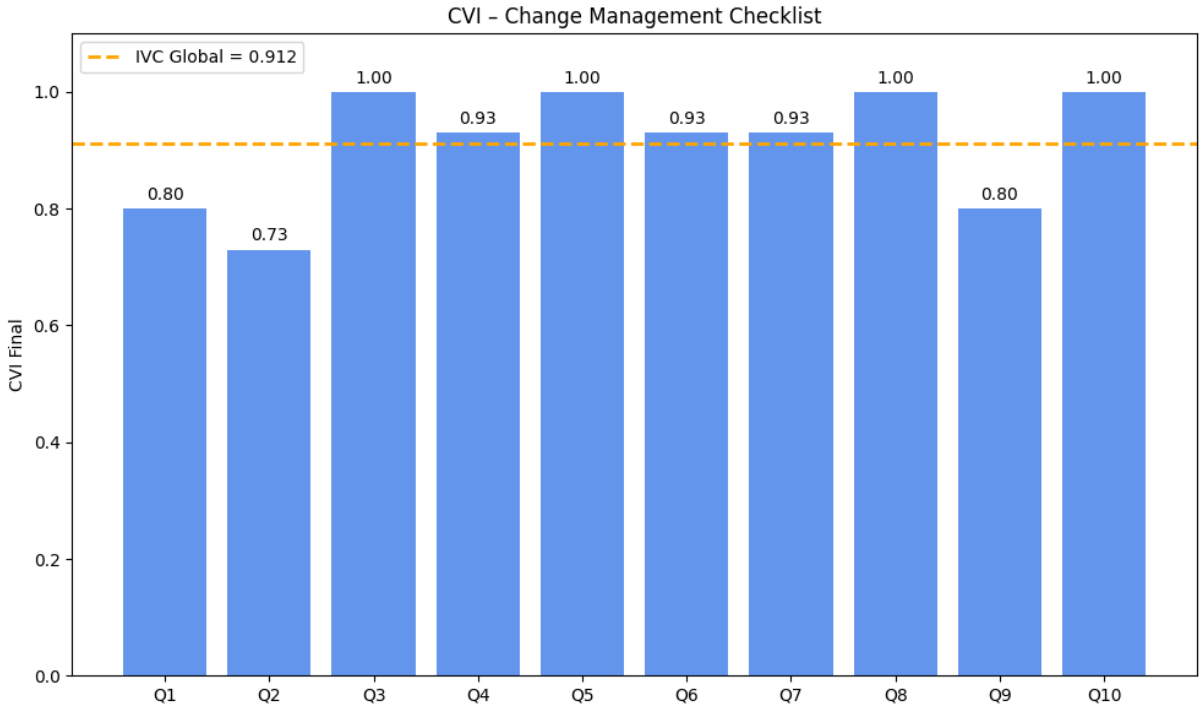
5.4.4 Validation Results - Change Management Checklist

The Change Management Checklist was evaluated by five professionals. Each of the ten questions was assessed using the 4-point Likert scale, focusing on the criteria of Clarity, Relevance, and Adequacy, as described earlier.

As shown in Figure 5.4, the majority of the items achieved a CVI ≥ 0.80 across all dimensions. Specifically, four items (Q1, Q4, Q6, and Q7) presented CVIs ranging between 0.80 and 0.93, while five items (Q3, Q5, Q8, Q9, and Q10) obtained the maximum possible score (CVI = 1.00) in all three criteria.

However, item Q2 received a lower rating in the Relevance and Adequacy dimensions, with a CVI of 0.60 in both, resulting in a final CVI of 0.73. According to Lynn's criteria [56], this score does not meet the minimum acceptable threshold of 0.78. Therefore, this item should be revised to improve its formulation and ensure its alignment with the checklist's purpose.

Figure 5.4: CVI Final by Question - Change Management Checklist



Despite the need for revision of this single item, the global CVI for the Change Management Checklist was calculated as **0.912**, indicating a high overall level of content validity. These results suggest that the checklist is well-structured, relevant, and applicable in agile project environments, and can serve as a practical tool to guide the evaluation and management of change, with minor adjustments to one specific element.

5.4.5 Qualitative Feedback from Experts

In addition to the quantitative evaluation using the Content Validity Index (CVI), the participating professionals were invited to provide open-ended feedback on each question and on the proposed checklists overall. These qualitative insights proved valuable for improving the clarity, scope, and applicability of the artifacts. Below is a synthesis of the main suggestions and comments, organized by checklist.

Scope Definition Checklist

Several suggestion highlighted the need for greater conceptual clarity in the questions:

- **Q1**: One reviewer noted that the term "main deliverable" could be more clearly defined, while another questioned whether the item properly related to the scope dimension.
- **Q2**: A suggestion was made to be more explicit about the consequences of not defining out-of-scope elements.
- **Q3**: A reviewer recommended including a brief definition of "stakeholders" to avoid confusion with end-users.
- **Q4**: While one evaluator agreed on the importance of defining benefits, they noted that some may not be foreseeable early in the project.
- **Q5 and Q6**: There were suggestions to split these into multiple questions, distinguishing technical from legal constraints and separating interfaces from integrations.
- **Q7**: Concerns were raised about the inclusion of quality requirements in the scope domain, and the use of generic terms like "other".
- **Q8**: Suggested to clarify what is meant by "team" and to separate operational from economic feasibility.

Requirements Gathering Checklist

The requirements checklist was generally well-received, with most questions validated quantitatively, though some improvements were suggested:

- **Q1**: One expert emphasized the importance of not expecting absolute clarity in a requirement, due to its inherently subjective nature.
- **Q3**: The item was seen as containing too many ideas in one sentence, potentially confusing.

- **Q6 and Q9**: Recommended clarifying the definition of "stakeholders".
- **Q8**: A suggestion was made to include the notion of change history and evolution of the requirement.

Change Management Checklist

This checklist was overall positively evaluated, though some structural and terminological suggestions emerged:

- **Q1**: It was recommended to split the item into two distinct questions (real change vs. customization).
- **Q2**: One reviewer questioned whether distinguishing between functional and non-functional requirements is meaningful, as both are equally essential.
- **Q3 and Q5**: Suggested rewording and separating into more precise sub-items.
- **Q7 and Q9**: Definitions for acronyms like "PO" and terms like "stakeholders" were requested to avoid misinterpretation.
- **Q8**: One evaluator praised the question's clarity; another recommended rewriting it for more direct phasing.
- **Q10**: It was proposed to separate the question into two, focusing on the nature and justification of resistance.

General Comments

The experts also offered broader recommendations to improve the artifacts:

- Include a glossary or introductory definitions for key terms such as "stakeholders", "scope", "change request", and "PO".
- Avoid compound questions - several items were identified as having more than one idea in a single sentence.

- Add new item addressing:
  - **Change-related risks** (technical, business).
  - **Rollback or contingency plans**.
  - **Regulatory or legal impacts**, especially in sensitive domains such as health and data privacy.
  - **Conflicts or inconsistencies between requirements**.
- Ensure that the checklist remains applicable in real-world settings and is not perceived as too abstract or idealized.

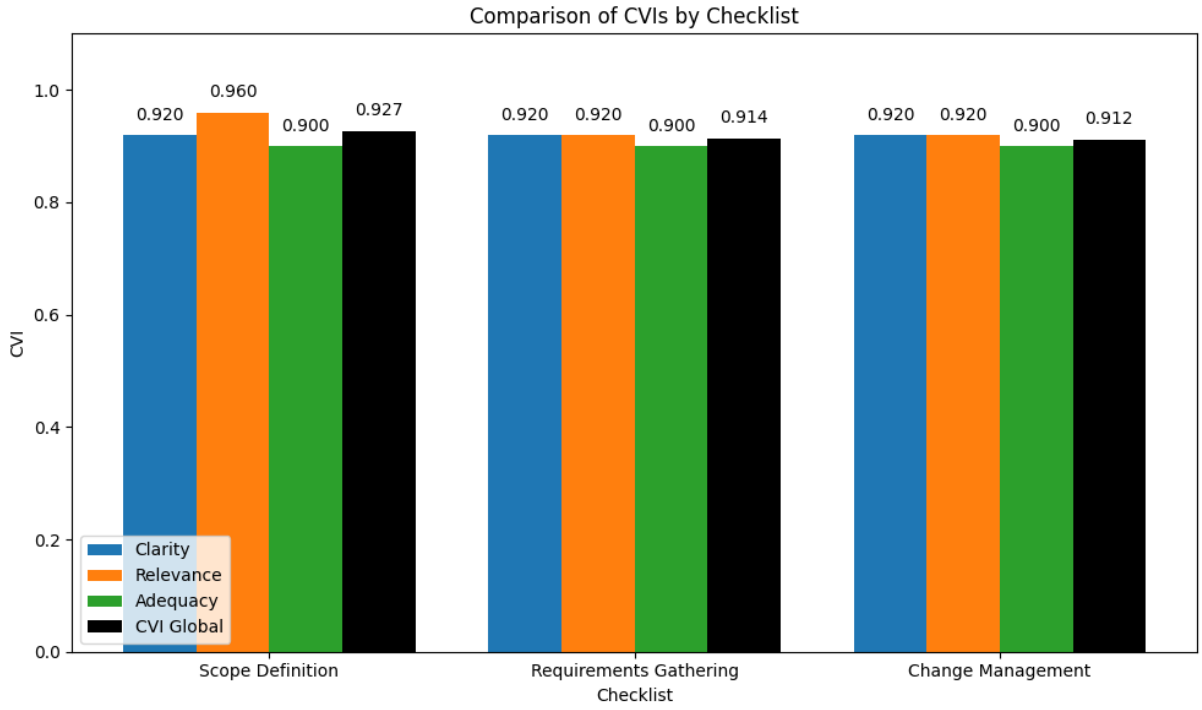
General Discussion and Summary of Checklist Validation

The validation of the three proposed checklists - Scope Definition, Requirements Gathering, and Change Management - confirmed their robustness and practical relevance in supporting change management within agile software development projects. This conclusion is based on both quantitative analysis, using the Content Validity Index (CVI), and qualitative feedback from experienced professionals.

Quantitatively, all three checklists achieved a **global CVI score above 0.90**, minimum threshold of 0.78 recommend by Lynn (1986) [56], and indicating strong consensus among experts regarding the clarity, relevance, and adequacy of the items.

- The **Scope Definition Checklist** obtained the highest global CVI (0.927), with most items receiving maximum scores, though minor concerns were raised about conceptual distinctions and compound questions.
- The **Requirement Gathering Checklist** followed with a global CVI of 0.914. One item (Q4) fell below the acceptable threshold in the clarity criterion, suggesting the need for refinement.
- The **Change Management Checklist** achieved a global CVI of 0.912, with one item (Q2) also requiring review due to divergent interpretations of its relevance and adequacy.

Figure 5.5: Comparative CVI Scores by Checklist



In addition to the numerical results, qualitative feedback enriched the evaluation. Reviewers emphasized the need for clear terminology, especially for recurring terms such as "stakeholders", "main deliverable", and acronyms like "PO". There was consensus on the importance of avoiding double-barreled questions, as several items were perceived to contain multiple ideas, potentially leading to misinterpretation. Reviewers also recommended the inclusion of new items addressing risk, contingency planning, regulatory impacts, and requirements conflicts, which are critical to comprehensive change assessment in software projects.

To visually reinforce the comparison, Figure 5.5 presents a consolidated view of the CVI scores across the three checklists, highlighting their strong performance and the consistency of validation.

Taken together, these results validate the proposed checklists as theoretically grounded, empirically supported, and ready for practical application in agile environments. The high CVI values indicate content quality and relevance, while the expert feedback provides a

valuable roadmap for future improvements and contextual adaptations.

Chapter 6

Final Considerations

This study aimed to investigate and propose strategies to improve change management in software development projects that adopt agile methodologies. From a combination of a systematic literature review, an empirical survey with industry professionals, and a validation process with expert reviewers, a structured approach was developed and evaluated. This approach is composed of three checklists: one for scope definition, one for requirements gathering, and another for change request evaluation, in addition to supporting templates.

The results showed that the checklists are well-aligned with the literature and practical needs of agile environments. The content validation, based on Lynn's method (1986) [56], demonstrated that most of the items were written, relevant, and adequate for their intended purpose. All three checklists obtained **global Content Validity Index (CVI) scores above 0.90**, indicating strong acceptance by experts in software engineering and project management.

Additionally, the integration of the checklist with the Scrum lifecycle showed that it can be embedded in agile ceremonies such as planning, refinement, and review, supporting structured decision-making without compromising agility. The proposed flow also highlighted how roles like Product Owner, Scrum Master, Developers, QA, and stakeholders interact with the instruments at each stage of the change process.

The qualitative feedback provided by the evaluators also brought valuable insights

for improving the instruments. Suggestions included the refinement of certain terms, the removal of compound questions, and the addition of items addressing risk, regulatory concerns, and contingency planning. These contributions reaffirm the relevance of collaborative validation in academic and practical research.

Overall, this research contributes to bridging the gap between theory and practice in agile change management by offering concrete and adaptable tools that promote clarity, traceability, and shared responsibility and decision-making.

6.1 Future Work

Although the checklists have been positively validated in terms of content, future work is necessary to ensure their effectiveness in a real-world setting and promote their broader adoption. The following initiatives are proposed as part of the next steps of this research:

- **Pilot Application in Real Projects**: Apply and monitor the use of the checklists in actual software development projects, both in industry and academia. This will allow for the evaluation of usability, practical utility, and impact on project success metrics (e.g., quality, cost, and schedule control).
- **Iterative Refinement Based on Field Feedback**: After applying the checklists in real contexts, collect qualitative and quantitative feedback from practitioners and teams to refine and improve the tools iteratively.
- **Tool Support and Automation**: Develop a digital or integrated version of the checklists to facilitate adoption and routine use in agile workflows.

Bibliography

[1] S. Ambler. "Agile requirements change management." (2014), [Online]. Available: http://agilemodeling.com/essays/changeManagement.htm.

[2] K. Beck, M. Beedle, A. V. Bennekum, *et al.* "Manifesto for agile software development." (2001), [Online]. Available: https://agilemanifesto.org/.

[3] A. Sellami and et al., "Towards an assessment tool for controlling functional changes in scrum process," in *Proceedings of the International Conference on Software Engineering*, Cited by: 4, 2018, pp. 34–47. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85053788563&partnerID=40&md5=2c55a235c4b4f5e8de433fa9bb867ec7.

[4] J. M. Hiatt, *ADKAR: A Model for Change in Business, Government and Our Community*. Prosci Learning Center Publications, 2006.

[5] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 9th. McGraw-Hill Education, 2020.

[6] I. Sommerville, *Software Engineering*, 9th. Addison-Wesley, 2011.

[7] W. W. Royce, "Managing the development of large software systems," in *Proceedings of IEEE WESCON*, Los Angeles, USA, 1970, pp. 1–9.

[8] P. Kruchten, *The Rational Unified Process: An Introduction*, 3rd. Addison-Wesley, 2004.

[9] F. Aizaz and et al., "An empirical investigation of factors causing scope creep in agile global software development context: A conceptual model for project managers," *IEEE Access*, vol. 9, pp. 109 166–109 195, 2021.

[10] K. Batool and I. Inayat, "An empirical investigation on requirements change management practices in pakistani agile based industry," in *Proceedings of the 2019 International Conference on Frontiers of Information Technology (FIT 2019)*, Accessed: 2024-09-02, IEEE, 2019, pp. 7–12. DOI: 10.1109/FIT47737.2019.00012. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85080123691&doi=10.1109%2fFIT47737.2019.00012&partnerID=40&md5=4f293a205d401f860541c743a996d2c7.

[11] N. Saher, F. Baharom, and O. Ghazali, "Requirement change taxonomy and categorization in agile software development," in *IEEE International Conference on Engineering Education*, Cited by: 16, 2017, pp. 1–6. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85050807226&doi=10.1109%2fICEEI.2017.8312441&partnerID=40&md5=7096eedd1815f8e35e5517fc7b4341c5.

[12] C. Berger and B. Rumpe, "Supporting agile change management by scenario-based regression simulation," *IEEE Transactions on Intelligent Transportation Systems*, pp. 504–509, 2010.

[13] *Itil 4 change enablement*, ITIL 4.

[14] IEEE Computer Society, *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, Version 3.0. IEEE, 2014.

[15] R. S. Pressman, *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 2019.

[16] K. Schwaber and J. Sutherland, *The scrum guide: The definitive guide to scrum*, Disponível em: https://scrum.org, 2017.

[17] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," no. EBSE 2007-001, 2007, Accessed: 2024-09-02. [Online]. Available: `https://legacyfileshare.elsevier.com/promis_misc/525444systematicreviewsguide.pdf`.

[18] *ITIL Foundation: ITIL 4 Edition*. Axelos Global Best Practice, 2019, ITIL 4 Change Enablement.

[19] K. Conboy and B. Fitzgerald, "Toward a conceptual framework of agile methods: A study of agility in different disciplines," *ACM*, 2004.

[20] D. F. Rico and H. H. Sayani, "The business value of agile software methods," *Journal of Information Systems Management*, 2009.

[21] S. Keaveney and K. Conboy, "Cost estimation in agile development projects," in *European Conference on Information Systems*, 2006.

[22] T. C. Lethbridge and R. Laganière, *Object-Oriented Software Engineering: Practical Software Development using UML and Java*. McGraw-Hill, 2001.

[23] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering*, 2nd. Prentice Hall, 2003.

[24] T. Maziaszek and H.-Y. Liong, "Requirements engineering for quality attributes," in *12th IEEE International Requirements Engineering Conference*, 2004.

[25] R. S. Pressman, *Engenharia de Software*, 7th ed. McGraw-Hill, 2011.

[26] P. M. Institute, *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*, 7th. Project Management Institute, 2021.

[27] R. S. Pressman, *Engenharia de Software*, 5th ed. McGraw-Hill, 2002.

[28] D. Bell, *Software Engineering: A Programming Approach*, 3rd ed. Addison-Wesley, 2000.

[29] L. H. Putnam and W. Myers, "How solved is the cost estimation problem?" *IEEE Software*, vol. 14, no. 6, pp. 88–91, 1997.

[30] B. Boehm and P. N. Papaccio, "Understanding and controlling software costs," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, pp. 1462–1477, 1988.

[31] *Ieee standard for software quality assurance processes*, 2014.

[32] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*. Springer, 2013.

[33] P. W. Stone, "Popping the (pico) question in research and evidence-based practice," *Applied Nursing Research*, vol. 15, no. 3, pp. 197–198, Aug. 2002, Accessed: 2024-09-02.

[34] A. Fogarty and et al., "Agile software development – do we really calculate the costs? a multivocal literature review," vol. 1251, pp. 203–219, 2020, Cited by: 5. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85089722961&doi=10.1007%2f978-3-030-56441-4_15&partnerID=40&md5=a4e41f3d970edceada33999e0bca869f.

[35] M. Alawairdhi, "Agile development as a change management approach in software projects: Applied case study," in *Proceedings of 2016 International Conference on Information Management, ICIM 2016*, Cited by: 7, 2016, pp. 100–104. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84978100590&doi=10.1109%2fINFOMAN.2016.7477541&partnerID=40&md5=fa30f4c58334dbccb1a1ee354eb03431.

[36] D. Albuquerque and et al., "Defining agile requirements change management: A mapping study," in *Proceedings of the ACM Symposium on Applied Computing*, Cited by: 7, 2020, pp. 1421–1424. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85083025107&doi=10.1145%2f3341105.3374095&partnerID=40&md5=a802146ba21f77075254b13de78eb099.

[37] S. Raza and U. Waheed, "Managing change in agile software development: A comparative study," in *2018 IEEE 21st International Multi-Topic Conference (INMIC)*, [S.l.: s.n.]: IEEE, 2018, pp. 1–8.

[38] S. Keaveney and K. Conboy, "Cost estimation in agile development projects," in *Proceedings of the 14th European Conference on Information Systems (ECIS)*, Cited by: 17, 2006. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84870639499&partnerID=40&md5=416504817d69e7e81e0762eabeaa4aa6.

[39] N. Nurmuliani, D. Zowghi, and S. Powell, "Analysis of requirements volatility during software development life cycle," in *Proceedings of the Australian Software Engineering Conference (ASWEC)*, Accessed: 2024-09-02, IEEE, 2004, pp. 28–37.

[40] R. E. Fairley, *Managing and Leading Software Projects*. Hoboken, NJ: Wiley-IEEE Computer Society Press, 2009, Accessed: 2024-09-02.

[41] N. Saher, F. Baharom, and R. Romli, "Identification of sustainability characteristics and sub-characteristics as non-functional requirement for requirement change management in agile," *International Journal of Scientific and Technology Research*, vol. 9, no. 3, pp. 5727–5733, 2020, Cited by: 1. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85082924414&partnerID=40&md5=77cfc8cfa4d5112e28ca59bb78023aea.

[42] M. A. Akbar and et al., "Az-model of software requirements change management in global software development," in *2018 International Conference on Computing, Electronic and Electrical Engineering (ICE Cube)*, 2018, pp. 1–6.

[43] K. Asad and M. Muqeem, "Enhancing requirements change request categorization and prioritization in agile software development using analytic hierarchy process (ahp)," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 11, no. 5, pp. 148–159, 2023, Cited by: 0, All Open Access, Gold Open Access. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85164618964&doi=10.17762%2fijritcc.v11i5.6589&partnerID=40&md5=fa5067f86d7d2a5c8a6c319a24084f1e.

[44] A. Law and S. Learn, "Waltzing with changes," in *Agile Development Conference (ADC'05)*, IEEE, 2005, pp. 279–285.

[45] S. Raza and U. Waheed, "Managing change in agile software development: A comparative study," in *2018 IEEE 21st International Multi-Topic Conference (INMIC)*, IEEE, 2018, pp. 1–8.

[46] Z. Shehzadi and et al., "A novel framework for change requirement management (crm) in agile software development (asd)," in *ACM International Conference Proceeding Series*, Cited by: 7, 2019, pp. 22–26. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85074280180&doi=10.1145%2f3357419.3357438&partnerID=40&md5=e7fde303f1fcd537a26b263cc924ce58.

[47] N. Saher, F. Baharom, and R. Romli, "Identification of sustainability characteristics and sub-characteristics as non-functional requirement for requirement change management in agile," *International Journal of Scientific and Technology Research*, vol. 9, no. 3, pp. 5727–5733, 2020, Accessed: 2024-09-02. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85082924414&partnerID=40&md5=77cfc8cfa4d5112e28ca59bb78023aea.

[48] A. Law and S. Learn, "Waltzing with changes [agile software development]," in *Agile Development Conference (ADC'05)*, 2005, pp. 279–285.

[49] M. Radović Marković, "Managing the organisational change and culture in the age of globalisation," *Journal of Business Economics and Management*, Aug. 2010, Accessed: 2024-09-02.

[50] Z. Ahmad, M. Hussain, A. Rehman, U. Qamar, and M. Afzal, "Impact minimization of requirements change in software project through requirements classification," in *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication (IMCOM '15)*, New York, NY, USA: Association for Computing Machinery, 2015. DOI: 10.1145/2701126.2701191.

[51] H. Ahmed, A. Hussain, and F. Baharom, "Current challenges of requirement change management," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 8, no. 10, pp. 173–176, 2016.

[52] H. Hakim, A. Sellami, and H. B. Abdallah, "An in-depth requirements change evaluation process using functional and structural size measures in the context of agile software development," in *ICSOFT 2020 - Proceedings of the 15th International Conference on Software Technologies*, Cited by: 2, 2020, pp. 361–375. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85091788410&partnerID=40&md5=37c2853.

[53] V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 6, pp. 728–738, 1984.

[54] S. McGee and D. Greer, "A software requirements change source taxonomy," in *Proceedings of the 4th International Conference on Software Engineering Advances (ICSEA 2009), Includes SEDES 2009: Simposio para Estudantes de Doutoramento em Engenharia de Software*, 2009, pp. 51–58.

[55] M. Bano, S. Imtiaz, N. Ikram, M. Niazi, and M. Usman, "Causes of requirement change - a systematic literature review," in *IET Seminar Digest*, 2011, pp. 22–31.

[56] M. R. Lynn, "Determination and quantification of content validity," *Nursing Research*, vol. 35, no. 6, pp. 382–385, 1986.

Appendix A

Scope Definition Checklist

Scope Definition Checklist

Project:

Responsible: Date:

Question	Compliance	Comments
1. Is the main deliverable clearly defined, including objectives and core functionalities?	☐	
2. Are out-of-scope elements identified to avoid ambiguity?	☐	
3. Have the stakeholders and end-users been identified, and are their needs considered?	☐	
4. Are the expected benefits (economic, operational, or strategic) clearly stated?	☐	
5. Have all technical, legal, operational, financial, and timeline constraints been identified?	☐	
6. Are system integrations and required interfaces specified or planned?	☐	
7. Does the scope include performance, reliability, or other critical quality requirements?	☐	
8. Has the scope's technical, operational, and economic feasibility been evaluated by the team?	☐	
9. Is the scope aligned with business goals and validated by stakeholders?	☐	
10. Are there measurable criteria to define project success and acceptance?	☐	

Appendix B

Requirements Gathering Checklist

Requirements Gathering Checklist

Project:

Responsible: Date:

Question	Compliance	Comments
1. Is the requirement written clearly and unambiguously?	☐	
2. Is the requirement understandable by all stakeholders and the technical team?	☐	
3. Is the requirement realistic in relation to current technology and within the project scope?	☐	
4. Does the requirement avoid anticipating implementation decisions (premature design)?	☐	
5. Is the requirement testable (can it be verified through testing or validation)?	☐	
6. Is the requirement necessary and does it meet the real need of the stakeholders?	☐	
7. Is the requirement traceable to its business objective or origin?	☐	
8. Does the requirement have flexibility for changes, if necessary?	☐	
9. Is the requirement accepted by all involved stakeholders?	☐	
10. Does the requirement directly contribute to the project or product objectives?	☐	

Appendix C

Change Management Checklist

Change Management Checklist

Project:

Responsible: Date:

Question	Compliance	Comments
1. Is the request a real change or a specific customization?	☐ Real Change	
2. Does the change refer to a Functional Requirement (RF) or a Non-Functional Requirement (NFR)?	☐	
3. Does the change impact existing requirements (scope, deliverables, or previous versions)?	☐	
4. Is there a direct impact on the schedule, cost, or team effort?	☐	
5. Will the change result in significant rework? If so, has it been assessed?	☐	
6. Has a technical impact and feasibility analysis been conducted?	☐	
7. Was the change discussed with the PO and/or Project Manager before the final decision?	☐	
8. Was the change recorded and prioritized in the backlog once accepted?	☐	
9. Does the change benefit more than one client or stakeholder, or bring collective value?	☐	
10. Was there resistance to change? If so, was it discussed and justified?	☐	

Template Change Request - Complete

Request ID:	
Change Requester:	Name of the person or team requesting the change
Approval Responsible:	Name of the PO, PM, or responsible stakeholder
Request Date:	Date the request was submitted
Affected System Version:	e.g., v1.2.0
Type of Change:	Bug fix, improvement, new feature, customization, other
Change Category:	FR (Functional Requirement), NFR (Non-Functional Requirement), interface, process, documentation, other
Detailed Description:	Explanation of the need and context
Change Impact	
Does it impact existing requirements?	Yes/No
Impacted or related requirements:	Indicate related FRs or NFRs
Does it impact the project scope?	Yes/No
Schedule Impact:	Describe estimated impact on timeline
Cost Impact:	Indicate if additional budget or funding is required
Team Effort Impact:	Estimated hours or additional resources needed
Technically Feasible?	Yes/No
Feasibility and Impact Analysis Notes:	Risks, dependencies, limitations, or alternative solutions evaluated
Acceptance Criteria:	How will we know the change has been implemented correctly?
Priority:	High/Medium/Low
Registered in the backlog:	Yes/No
Change Request Status:	Not started

Template Change Request - Simplified

Request ID:	

Change Requester:	Name of the person or team requesting the change
Approval Responsible:	Name of the PO, PM, or responsible stakeholder
Request Date:	Date the request was submitted
Affected System Version:	e.g., v1.2.0
Type of Change:	Bug fix, improvement, new feature, customization, other
Change Category:	FR (Functional Requirement), NFR (Non-Functional Requirement), interface, process, documentation, other
Detailed Description:	Explanation of the need and context
Change Impact:	Scope/Schedule/Cost
Does it impact existing requirements?	Yes/No
Impacted or related requirements:	Indicate related FRs or NFRs
Technically Feasible?	Yes/No
Acceptance Criteria:	How will we know the change has been implemented correctly?
Priority:	High/Medium/Low
Change Request Status:	Not started

Appendix D

Survey

Survey on Change Management in Software Project

Welcome to our survey! Your participation will help propose improvements and best practices in management change. Completing it will take approximately 10 minutes.

Section 1 – Instructions and Information

This questionnaire aims to collect data for an academic analysis of how change management is currently applied and handled in software systems.

The survey is part of a master's dissertation developed by Kamila Antunes, a Software Engineering graduate from the Federal University of Technology – Paraná (UTFPR), Dois Vizinhos campus, Brazil, and currently a Master's student in Computer Science at the Polytechnic Institute of Braganza (IPB), Portugal.

By participating in this research, you are contributing to the development of a guide to gain better understanding of the impact of change management on software projects.

- **Confidentiality**: The survey is anonymous, and your responses will not be identified.
- **Voluntary Participation**: Participation is entirely voluntary, and you may withdraw at any time.

If you are interested in receiving the results of this study after its conclusion, please provide your email in the designated field at the end of the form.

For questions or additional information, feel free to contact Kamila Antunes through the following channels:

- **Email**:
  - kamilaantunes1@gmail.com
  - kamilaneves@alunos.utfpr.edu.br
  - a61441@alunos.ipb.pt
- **LinkedIn**: https://www.linkedin.com/in/kamila-antunes/

We deeply appreciate your time and valuable contribution to this research!

1. Have you read the instructions provided and information and agree to participate in this research?
   - ☐ Yes, I agree to participate and proceed with the questionnaire.
   - ☐ No, I do not wish to participate.

Section 2 – General Information

2. Current role:
   - ☐ Developer
   - ☐ Project Manager
   - ☐ Product Owner

- Scrum Master
- Tech Leader
- Teacher/ Researcher
- Other

3. Seniority level:
    - Junior
    - Mid-Level
    - Senior
    - Specialist/ Leader
    - Executive/ C-Level
4. Academic background:
    - Incomplete higher education
    - Completed higher education
    - Postgraduate/ MBA
    - Master’s degree
    - Doctorate
5. Country of residence
    - Brazil
    - Portugal
    - Other
6. Organization’s country/ State or region
7. Organization size
    - Small company (up to 50 employees)
    - Medium-sized company (51-500 employees)
    - Large company (more than 500 employees)
8. Industry
    - Education
    - Financial
    - Manufacturing/ Industry
    - Healthcare
    - Information Technology/ Software
    - Other
9. Average number of projects managed to date
    - 1-5 projects
    - 5-10 projects
    - 10-20 projects
    - More than 20 projects
10. Years of experience in Project Management
    - Less than 1 year
    - 1-3 years
    - 3-5 years
    - 5-10 years

- More than 10 years

11. Years of experience in Change Management
    - No experience in change management
    - Less than 1 year
    - 1-3 years
    - 3-5 years
    - 5-10 years
    - More than 10 years
12. Experience with Change Management in projects
    - Very frequently
    - Frequently
    - Occasionally
    - Rarely
    - Never
13. Academic experience (involvement in teaching/research)
    - None
    - Some experience as a student-researcher
    - Experience as an active teacher/researcher
    - Extensive experience in academic environments
14. Do you currently use Agile Methodology?
    - Yes (specify which)
    - No
15. If you answered “Yes” to the previous question, please specify the methodology:

Section 3 – Change Management Practices in Software Projects

16. How is the impact analysis of project changes (requirements, costs, and/or schedule) conducted?
    - Formal analysis involving the entire team
    - Analysis led by the Product Owner or Project Manager
    - Other
17. Is a margin of error in the schedule previously considered to handle changes?
    - Very frequently
    - Frequently
    - Occasionally
    - Rarely
    - Never
18. What types of changes are most frequent in your projects?
    - Functional requirements changes
    - Non-functional requirements changes
    - Technology changes
    - Project team changes
    - Other

19. Which types of changes are most likely to compromise project's schedule, budget, or quality?
    - ☐ Functional requirements changes
    - ☐ Non-functional requirements changes
    - ☐ Technology changes
    - ☐ Project team changes
20. Why do you think these types of changes have such an impact?
21. What impact are most frequently observed due to changes?
    - ☐ Cost
    - ☐ Schedule
    - ☐ Scope
    - ☐ Rework
    - ☐ Quality
    - ☐ Workflow
22. Do changes during the project usually compromise the final product quality?
    - ☐ Very frequently
    - ☐ Frequently
    - ☐ Occasionally
    - ☐ Rarely
    - ☐ Never
23. How do you rate the impact of changes on software project quality?
    - ☐ Significantly improves
    - ☐ Improves
    - ☐ Does not interfere
    - ☐ Reduces
    - ☐ Significantly reduces, compromising the project
24. How are internal changes (not requested by clients/stakeholders) communicated to them?
    - ☐ They are not communicated
    - ☐ Through formal meetings
    - ☐ Only changes with significant impact are communicated
25. How are changes tracked throughout the project lifecycle?
    - ☐ They are not controlled
    - ☐ Through formal control processes
    - ☐ Continuous backlog adjustments using an agile approach
    - ☐ Other
26. Are change requests assessed for effort and impact (cost/schedule) before being accepted?
    - ☐ Very frequently
    - ☐ Frequently
    - ☐ Occasionally
    - ☐ Rarely

- Never

27. Do high-impact changes face resistance in your organization?
    - Yes
    - No, they are generally accepted
    - No, they are immediately rejected
28. If your previous answer was “Yes”, could you please describe how these changes are handled?
29. How often are changes prioritized before being included in the backlog?
    - Very frequently
    - Frequently
    - Occasionally
    - Rarely
    - Never
30. How are requirement changes integrated into the backlog?
    - Backlog review to identify possible increments
    - Treated as independent requests
    - Depends on complexity
31. How are changes tracked throughout the project?
    - Manual documents
    - Projects management tools (e.g., Jira, Trello)
    - They are not tracked
    - Other
32. How is the change approval process conducted?
    - Direct decision by the Product Owner/ Project Manager
    - Discussion in meetings with impact estimates before the decision
    - Changes requested by stakeholders are accepted without impact analysis
    - Other
33. Who is primarily responsible for accepting/rejecting project changes?
    - Clients/ Stakeholders
    - Technical team (developers, designers)
    - Project Manager
    - Product Owner
    - Director
    - Other

Section 4 – Impacts and Recommendations

34. Changes most frequently result in:
    - Schedule overrun
    - Schedule reduction
    - Neither, the schedule remain the same
35. Impact of change on costs:
    - Cost increase

- Cost reduction
- Costs remain stable

36. Main challenges faced when implementing changes:
    - Schedule adjustments
    - Unexpected cost increases
    - Impact on quality
    - Other
37. Perception of product quality improvement after changes:
    - Very frequently
    - Frequently
    - Occasionally
    - Rarely
    - Never

Section 5 – Recommendations to improve change management in software?

38. In terms of **quality**:
39. In terms of **schedule**:
40. In terms of **cost**:

Section 6 – Final

Thank you for your time and participation!

41. As mentioned at the beginning, if you are interested in receiving the results of this work upon completion, please provide your email:

Appendix E

Checklist Evaluation Form by Experts

Checklist Evaluation Form by Experts

1. Evaluator Information

Name: _____

Academic Background: _____

Area of Expertise: _____

Years of Experience in the Field: _____

2. Checklist Evaluation

We kindly request your collaboration in evaluating three checklists developed within the context of a research study on change management in software development projects. The aim of the study was to understand the main impacts of changes in this type of project. Based on results obtained through a systematic literature review and a survey with professionals in the field, practical tools were developed to support the following:

- Assessment of maturity in project scope definition
- Validation of function and non-functional requirements
- Support for managing change requests throughout the project life cycle

Your participation is essential to validate the clarity, relevance, and appropriateness of the proposed items, directly contributing to the rigor and applicability of the research. Your critical analysis will be extremely valuable for improving both the study and the instruments.

Please evaluate each checklist item based on the following criteria:

- **Clarity** – Is the item written in an understandable way?
- **Relevance** – Is the item important for the checklist's objective?
- **Adequacy** – Is the item properly formulated for its intended purpose?

Scope Definition					
ID	Checklist Item	**Clarity** Is the item written in an understandable way?	**Relevance** Is the item important for the checklist's objective?	**Adequacy** Is the item properly formulated for its intended purpose?	Comments / Additional Suggestions
1	Is the main deliverable clearly defined, including objectives and core functionalities?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
2	Are out-of-scope elements identified to avoid ambiguity?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
3	Have the stakeholders and end-users been identified, and are their needs considered?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
4	Are the expected benefits (economic, operational, or strategic) clearly stated?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
5	Have all technical, legal, operational, financial, and timeline constraints been identified?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
6	Are system integrations and required interfaces specified or planned?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
7	Does the scope include performance, reliability, or other critical quality requirements?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
8	Has the scope's technical, operational, and economic feasibility been evaluated by the team?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	

9	Is the scope aligned with business goals and validated by stakeholders?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
10	Are there measurable criteria to define project success and acceptance?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	

Requirements Gathering					
ID	Checklist Item	**Clarity** Is the item written in an understandable way?	**Relevance** Is the item important for the checklist's objective?	**Adequacy** Is the item properly formulated for its intended purpose?	Comments / Additional Suggestions
1	Is the requirement written clearly and unambiguously?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
2	Is the requirement understandable by all stakeholders and the technical team?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
3	Is the requirement realistic in relation to current technology and within the project scope?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
4	Does the requirement avoid anticipating implementation decisions (premature design)?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
5	Is the requirement testable (can it be verified through testing or validation)?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
6	Is the requirement necessary and does it meet	() Strongly disagree () Partially disagree	() Strongly disagree () Partially disagree	() Strongly disagree () Partially disagree	

	the real need of the stakeholders?	() Partially agree () Strongly agree	() Partially agree () Strongly agree	() Partially agree () Strongly agree	
7	Is the requirement traceable to its business objective or origin?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
8	Does the requirement have flexibility for changes, if necessary?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
9	Is the requirement accepted by all involved stakeholders?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
10	Does the requirement directly contribute to the project or product objectives?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	

Change Management					
ID	Checklist Item	**Clarity** Is the item written in an understandable way?	**Relevance** Is the item important for the checklist's objective?	**Adequacy** Is the item properly formulated for its intended purpose?	Comments / Additional Suggestions
1	Is the request a real change or a specific customization?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
2	Does the change refer to a Functional Requirement (RF) or a Non-Functional Requirement (NFR)?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
3	Does the change impact existing requirements (scope, deliverables, or previous versions)?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	

4	Is there a direct impact on the schedule, cost, or team effort?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
5	Will the change result in significant rework? If so, has it been assessed?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
6	Has a technical impact and feasibility analysis been conducted?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
7	Was the change discussed with the PO and/or Project Manager before the final decision?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
8	Was the change recorded and prioritized in the backlog once accepted?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
9	Does the change benefit more than one client or stakeholder, or bring collective value?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	
10	Was there resistance to change? If so, was it discussed and justified?	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	() Strongly disagree () Partially disagree () Partially agree () Strongly agree	

3. General Comments

Please use the space below to record any criticisms, general suggestions, missing items, or relevant observations: