Proof of Concept in Software Engineer

by

Bruno Fernando ANTOGNOLLI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN SOFTWARE ENGINEERING
M.A.Sc.

MONTREAL, DECEMBER 17, 2024

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

M. Fabio Petrillo, Thesis supervisor
Department of Software Engineering and IT, École de Technologie Supérieurel

Mrs. Imen Benzarti, Chair, Board of Examiners
Department of Software Engineering and IT, École de Technologie Supérieure

M. William de Paula Ferreira, Member of the Jury
Department of Systems Engineering, École de Technologie Supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON DECEMBER 13, 2024

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

FOREWORD

While PoCs are frequently utilized to address uncertainties and validate technological feasibility, the lack of a structured approach to their implementation often leaves practitioners navigating ambiguities. This work emerges from the necessity to bridge this gap, providing a clearer path toward systematic PoC processes that support informed decision-making in complex projects.

The absence of a well-defined PoC framework has highlighted the risks associated with inconsistent PoC practices. This thesis offers a structured approach, aiming to refine PoC definition, clarify processes, and present a framework that enhances the effectiveness and reliability of PoCs. It is my hope that this research contributes to a more standardized understanding of PoCs, fostering innovation, efficiency, and accountability across software engineering projects.

ACKNOWLEDGEMENTS

This thesis is dedicated to the memory of my grandparents, with special tribute to my maternal grandmother, Francisca Sabatel Martins, who was the last of them to pass away during the course of this work. Their legacy of resilience and dedication has been a profound source of inspiration. This work would not have been possible without the invaluable guidance and support of my advisor, Fabio Petrilo, whose expertise shaped the direction and rigor of this research. I am immensely grateful to my family and friends for their unwavering support and patience, which have been a constant source of strength throughout this journey. I also extend my heartfelt thanks to my colleagues in software engineering, whose encouragement and shared experiences enriched this research. Finally, I want to express my deep gratitude to Gisele Fatala for her love, support, and belief in me throughout this journey.

Cadre logiciel de preuve de concept en génie logiciel

Bruno Fernando ANTOGNOLLI

RÉSUMÉ

La preuve de concept (PoC) est une pratique adoptée en ingénierie logicielle, mais sa définition et sa mise en œuvre manquent souvent de clarté et de cohérence dans ce domaine. Cette dissertation examine le concept de PoC dans le but de fournir une définition et un cadre structuré pour son développement. À travers une revue systématique de la littérature académique et grise, nous identifions les caractéristiques clés, les processus et les meilleures pratiques associés aux PoC réussies. La recherche révèle une lacune dans la littérature académique concernant les descriptions détaillées du processus de PoC, soulignant ainsi la nécessité d'une compréhension plus approfondie. En synthétisant des informations issues de diverses sources, ce travail propose une définition affinée des PoC en ingénierie logicielle et esquisse un cadre qui englobe les étapes de planification, de mise en œuvre, de validation technique et de prise de décision. Cette recherche contribue au domaine en offrant une conceptualisation claire des PoC et en fournissant des conseils pratiques pour leur exécution efficace, améliorant ainsi le taux de réussite des projets logiciels.

Mots-clés: PoC, Preuve de concept, cadre logiciel, génie logiciel

Proof of Concept in Software Engineer

Bruno Fernando ANTOGNOLLI

ABSTRACT

The Proof of Concept (PoC) is a practice adopted in software engineering, yet its definition and implementation often lack clarity and consistency in the field. This dissertation investigates the concept of PoCs aiming to provide a definition and a structured framework for their development. Through a systematic review of academic and grey literature, we identify the key characteristics, processes, and best practices associated with PoCs. The research reveals a gap in the academic literature regarding detailed descriptions of the PoC process, highlighting the need for a more comprehensive understanding. By synthesizing insights from diverse sources, this work propose a refined definition of PoCs in software engineering and outline a framework that encompasses planning, implementation, technical validation, and decision-making stages. This research contributes to the field by offering a clear conceptualization of PoCs and providing practical guidance for their effective execution, ultimately enhancing the success rate of software projects.

Keywords: PoC, proof of concept, framework, software engineering

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS

PoC	Proof of Concept
IT	Information Technology
NASA	National Aeronautics and Space Administration
NSA	National Security Agency
KPI	Key Performance Indicator
Ngram	Google Ngram Viewer

INTRODUCTION

0.1 Context and motivations

The Proof of Concept (PoC) is widely used in the technology field for various purposes. PoCs help reduce risks and address unknowns that may pose challenges during project execution Baptista & Abbruzzese (2024), enabling quicker experimentation Ingeno (2018); Burns (2018); Enríquez & Salazar (2018); Beningo (2022); Tune & Perrin (2024). They also support learning by providing hands-on exploration of new technologies Morris (2016); Shrivastava, Srivastav, Sheth, Karmarkar & Arora (2022). Additionally, PoCs facilitate technology evaluation, guiding decision-making in complex environments Ford, Parsons & Kua (2017), Brisals & Hedger (2024).

The definition of PoC can be found in dictionaries as '*the stage during the development of a product when it is established that the product function as intended*' Collins (2024) or as "*evidence that shows that a business proposal, design idea, etc., will work, usually based on an experiment or a pilot project*" Oxford (2024). Although these definitions can be applied in many contexts, they do not fully capture the real meaning of a software proof of concept necessitating further refinement within the software engineering field.

Despite their widespread use in software engineering, there is a notable lack of a definition of PoC within this domain. To our knowledge, no substantial efforts have been made to systematize what constitutes a PoC in the IT field. Therefore, the current lack of a clear PoC definition and a process highlights a significant opportunity for improvement of PoC process in software engineering. This thesis seeks to address this gap by providing a refined definition of PoCs and establishing a structured framework. This comprehensive approach aims to propose a standard PoC process, understanding of its steps and facilitating more effective planning, execution, and evaluation of PoCs in software development.

0.2 Problem and research questions

The field of Software Engineering faces a significant challenge due to the conceptual ambiguity surrounding Proofs of Concept (PoCs). The terminology used to describe PoCs is often inconsistent, with terms like "PoC," "prototype," and "pilot" frequently being used interchangeably Larman (2002), Mistrik (2012), Van Hecke (2013). This lack of clarity makes it harder to set clear goals for PoCs and evaluate their success.

The lack of a clear definition for PoC in software engineering has led some authors to use the terms PoC, Prototype, and MVP synonymously, treating them as interchangeable Larman (2002); Fairbanks (2010); Mistrik (2012); Van Hecke (2013); Anderson (2023). However, others argue that these terms are distinct, highlighting differences in their scope and purpose Ingeno (2018); Shrivastava *et al.* (2022); Prateek (2024); TechBohdana (2024). This ambiguity has resulted in ad hoc practices that undermine the effectiveness of the PoC process.

The consequences of this conceptual gap extend throughout software development projects. For instance, misallocation of resources is a frequent problem Pearson (2024). Without well-defined objectives and evaluation criteria, teams may overinvest in PoC efforts that are misaligned with project goals or, conversely, underinvest in initiatives that hold high potential. This not only increases project risk but can also lead to costly delays or even outright failure when PoCs fail to validate hypotheses or mitigate risks. Furthermore, when the PoC process lacks clarity, opportunities for innovation may be missed. A poorly defined approach can stifle experimentation, limiting the potential for discovering new and innovative solutions Kupitman (2019).

To address these challenges, the next sections introduce the research questions that guide this study.

0.3 RQ1.: What is the definition of Proof of Concept (PoC) in the context of Software Engineering

This question addresses a gap in the current understanding of PoCs, which are widely used but often inconsistently defined. This question seeks to eliminate confusion and ambiguity surrounding PoCs, especially in relation to similar concepts such as prototypes and pilots. Addressing this question lays the groundwork for the development of a standardized framework that guides practitioners in the effective application of PoCs throughout the software engineering field. By answering this question, the thesis contribute to a more coherent understanding of PoCs and provide practitioners with a clear reference point.

0.4 RQ2.: What are the characteristics and processes that contribute to planning, execution, and evaluation of PoCs in software engineering

By identifying PoC characteristics and processes, this research provide actionable insights into how PoCs can be executed more effectively, reducing risks and verify the feasibility of the experiment. Moreover, integrating these insights into a comprehensive framework offers a structured approach for practitioners, ensuring that PoCs are consistently aligned with goals and evaluated against clear criteria.

0.5 RQ3.: What are the benefits and challenges of implementing a structured framework for Proof of Concept (PoC) execution in software engineering

This question explores the benefits and challenges of introducing a structured framework for PoC execution in software engineering. Building on the foundational elements established in RQ1 (the definition of PoC) and RQ2 (characteristics, processes, and practices of PoCs), RQ3 seeks to assess the feasibility and implications of creating a framework to guide practitioners.

Ultimately, the framework aspires to bridge the gap between theoretical insights and practical application, offering the software engineering community a valuable tool for the execution of PoCs.

0.6 Research Methodology

The research methodology is structured around three key phases to provide a comprehensive understanding of Proof of Concept (PoC) in the field of software engineering:

1. **Historical Review:** The first phase involves tracing the origins and early usage of the term PoC within the context of Software Engineering. This historical analysis helps establish the foundation for understanding how PoCs have evolved over time and their role in various technological advancements.
2. **Analysis of PoC Usage:** The second phase focuses on analyzing how different authors and researchers have applied PoCs in the software engineering domain. This analysis aims to identify the core components of the PoC process, including its characteristics, best practices, and challenges.
3. **Framework Development and Evaluation:** Based on the insights gathered from the historical review and literature analysis, a framework for creating more accurate and structured PoCs in software engineering is proposed. This framework is then evaluated through its application in real-world case studies or scenarios to validate its effectiveness and practicality in addressing PoC challenges.

This three-step approach ensures a thorough exploration of PoC development, combining historical context, practical analysis, and framework validation to contribute to a clearer understanding and implementation of PoCs in software engineering.

0.7 Contributions

This research aims to drive the gap by providing a clearer definition of PoC in software engineering and suggesting a framework for its effective development. Three approaches are used to achieve this.

First, at a theoretical level, this thesis aims to propose a definition of a PoC within the context of Software Engineering. By addressing the conceptual ambiguity that currently surrounds PoCs, this research contribute to a more consistent and shared understanding of PoCs, distinguishing it from similar concepts like prototypes and pilots.

Second, on a practical level, this thesis identifies the characteristics and processes of PoCs, establishing a well-defined approach for their planning, execution, and evaluation. By synthesizing these elements into a comprehensive framework, the research provide a practical guide for software engineers, project managers, and other practitioners. This framework serves as a structured guide for PoC practitioners, helping align PoC outcomes with initial goals, providing metrics to inform decision-making, and supporting the learning process.

Ultimately, the contributions of this thesis help bridge the gap between theory and practice in the use of PoCs, offering both conceptual insights and actionable strategies to improve the effectiveness of PoCs in software development. This work support the broader software engineering community in achieving greater project success, fostering innovation, and reducing the risks associated with poorly defined or executed PoCs.

0.8 Thesis Outline

This thesis is structured into several chapters, each addressing a critical aspect of the research on Proof of Concepts (PoCs) in software engineering. The chapters are organized as follows:

- **Introduction** This chapter provides an overview of the research context, including the motivations for studying PoCs in software engineering. It introduces the research problem, presents the key research questions, and outlines the contributions of the thesis to both theory and practice.
- **Chapter 1: Defining a PoC in Software Engineering** This chapter presents a historical overview of PoCs and their evolution, particularly within the field of software engineering. It discusses various definitions and characteristics of PoCs found in the literature and proposes a refined definition suited for the software engineering domain.
- **Chapter 2: PoC Process and Framework Development** This chapter develops a comprehensive framework for the planning, execution, and evaluation of PoCs in software engineering. It details each stage of the PoC process, including planning, technical validation, and decision-making, with the goal of providing practical guidance for industry practitioners.
- **Chapter 3 and Chapter 4: Application and Case Study** This chapter describes the application of the proposed PoC framework in a real-world software development project. It details the design and execution of the case study, assessing the effectiveness of the framework and identifying the challenges encountered during the implementation.
- **Chapter 5: Discussion** This chapter discusses the broader implications of the research findings for both the academic and professional communities. It evaluates the practical value of the proposed PoC framework and highlights areas for further research, including potential improvements to the framework.
- **Chapter 6: Conclusion** The final chapter summarizes the key contributions of the thesis, focusing on the importance of establishing a clear definition of PoCs in software engineering. It reiterates the practical benefits of the proposed framework and discusses its potential to enhance the success rate of software development projects.

CHAPTER 1

DEFINING A POC IN SOFTWARE ENGINEERING

It is important for a PoC practitioner to understand where the concept of proof of concept (PoC) in software engineering comes from—not only knowing its origins, but also how it has adapted within the field of software engineering. Since this concept did not originally emerge from software engineering, it has undergone adaptations over time. As a result of these changes, the concept has yet to be clearly defined in an academic context within software engineering, and there is still no consensus on its definition within the field.

This chapter begins by tracing the evolution of PoC, with a historical overview that highlights its roots in fields such as aerospace and biomedical engineering, before it was adopted by the software industry. Understanding the historical context is crucial for grasping how PoC has been adapted, allowing us to identify the factors that have influenced its ambiguous status in software engineering.

In addition, the chapter explores key debates surrounding the PoC concept, particularly the distinction between PoCs, prototypes, and minimum viable products (MVPs), as well as whether PoC code should be reused or discarded. These discussions underscore the importance of clarifying PoC's role and its implications for software development.

The chapter concludes by proposing a more refined definition of PoC within the context of software engineering. This new definition emphasizes PoC's experimental and short-term nature, differentiating it from other methods like prototyping and MVPs, and providing clearer guidance for its application in the field.

1.1 Historical Overview of PoC

Understanding the historical development of Proof of Concept (PoC) provides context for understand how the concept evolving across different domains and underscores the need for a clear understanding of its purpose and characteristics within software engineering.

The concept of Proof of Concept (PoC) originated in fields outside of software engineering, such as aerospace and biomedical industries, before being adapted to technological contexts Deckert & Szalai (1954); Welker (1963); Mun (1965). In the following decades, the concept gained broader acceptance and expanded into various fields. In the 1970s, public research institutions began to adopt and adapt PoC for their specific purposes. By the 1980s, the biomedical field also integrated PoC terminology into its practices. Furthermore, entrepreneurial communities started embracing PoC methodologies during the 1980s and early 1990s Jobin, Hooge & Le Masson (2020).

Figure 1.1 illustrates the frequency of the term "PoC" within the software engineering field over time. This data was retrieved using the Google Books Ngram Viewer, an online tool that charts the frequency of search strings from printed sources published between 1980 and 2019.

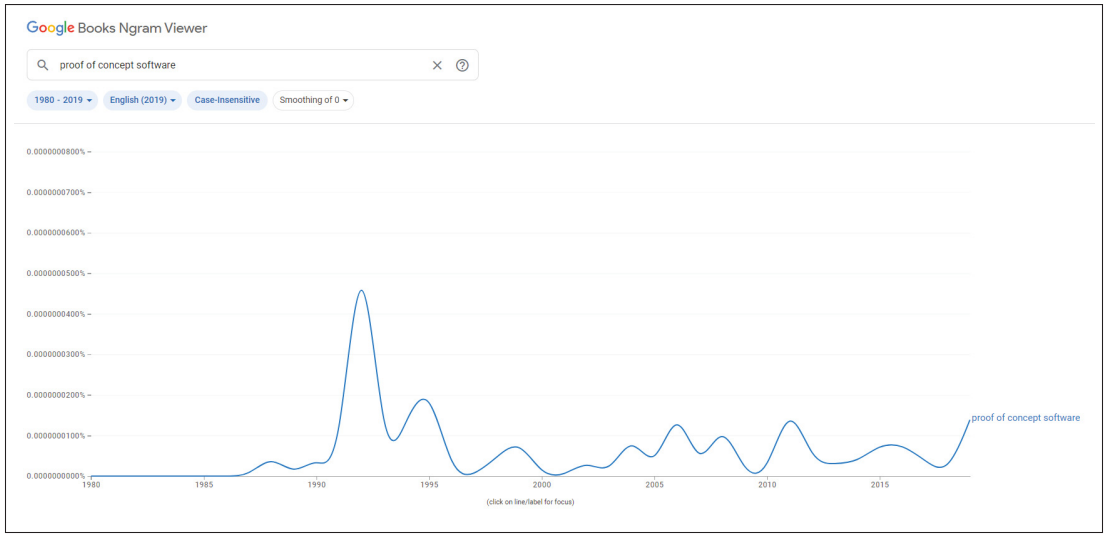


Figure 1.1 Use of the proof-of-concept term over time

PoC found its place in the information technology (IT) field during the 1980s and 1990s, coinciding with the major outsourcing of IT operations to firms like IBM. This period marked a pivotal moment when companies began relying on external expertise to validate and implement new technological concepts. Consequently, the PoC approach became aligned with the need for companies to test the feasibility of technological solutions before full implementation Jobin *et al.* (2020).

The first recorded mention of "proof of concept software" was in 1981, when NASA used the term to validate software modules for the Integrated Programs for Aerospace Vehicle Design (IPAD) Bales (1981). In 1985, NASA once again utilized PoC for testing new ideas in the IPAD project Fulton (1985). However, neither of these early references provided a definition of PoC, nor did they outline the specific characteristics required to develop one. Other references from this period pertain to PoC in different domains, which are beyond the scope of this study of New York at Binghamton. Department of Geography, of Geography & of Geography (1987).

In 1991, PoC was employed to validate concepts related to an Artificial Intelligence (AI) algorithm Society (1991). By 1993, PoC became a common method for evaluating new and emerging technologies, typically occurring in the presolicitation phase of a project. These PoCs primarily focused on pure research and were defined in relation to the specific problem being addressed on Governmental Affairs (1993).

The U.S. government also played a role in the development of PoC software. In 1994, it initiated efforts to enhance software robustness, user-friendliness, and readiness for commercial development. The goal was to facilitate the transfer of solutions to various industries on Science Space & Technology (1994). In the same year, the U.S. Committee on Science, Space, and Technology aimed to foster collaboration between universities, research labs, and system vendors through PoC development on Science. Subcommittee on Energy & Environment (1994).

In 1995, NASA utilized a PoC to develop and test integrated reliability analysis capabilities within a single tool, focusing on testing potential failures Aeronautics *et al.* (1995). Concurrently, the National Security Agency (NSA) developed a PoC in collaboration with two automation companies to reduce administrative costs on Governmental Affairs. Subcommittee on Oversight of Government Management & the District of Columbia (1995).

As PoCs continued to gain traction in various fields through the late 1990s, their application in software engineering became more widespread. This increasing use led to ongoing debates in the early 2000s about the distinction between "PoC" and "prototype." Some authors use the terms

interchangeably Larman (2002); Mistrik (2012); Van Hecke (2013), while others differentiate them. A key distinction lies in their purpose. PoCs aim to demonstrate the feasibility of a concept or technology, often with a limited features Anderson (2023). Prototypes, on the other hand, can have broader goals, such as validating design ideas or guiding software implementation Mistrik (2012). Moreover, PoCs are typically short-lived and not intended for final product use Ford *et al.* (2017); Burns (2018); Hinchey (2018), while prototypes may evolve into functional components.

The disposable nature of PoCs is a recurring theme in the literature. They are often regarded as quick experiments designed to assess technical feasibility Burns (2018). This concept aligns with Martin Fowler's notion of "sacrificial architecture" Fowler (2015), wherein PoC code is considered disposable due to design compromises made for rapid development Ford *et al.* (2017). While some argue that a PoC should never evolve into a deliverable due to the potential technical debt it accumulates Beningo (2022), others maintain that its purpose is solely to assess feasibility Ford *et al.* (2017); Burns (2018); Hinchey (2018).

1.2 Research Approach

The research methodology involves a structured approach, focusing on two key sources of information: academic papers and grey literature. Each source is reviewed through predefined inclusion and exclusion criteria, ensuring that only relevant studies are considered. This approach helps bridge the gap between studies provided by academic research and the more practical implementations found in grey literature.

For both types of sources, the criteria for inclusion required that studies explicitly address the software engineering domain, with a particular emphasis on the PoC process. Only studies that provided detailed descriptions of the PoC process, including steps, strategies, and decision-making process, were considered. For grey literature, additional scrutiny was applied to ensure the credibility of the sources.

1.2.1 Academic Literature

The search for academic literature was conducted using the Scopus database, focusing on research published in well-established software engineering journals and conferences.

For academic literature, specific inclusion criteria were established to maintain focus and relevance. First, studies or publications were required to explicitly address the software engineering domain, with particular emphasis on the PoC process or methodology. Only those articles that provided detailed descriptions of the PoC process, including insights into the specific steps, strategies, and decision-making processes involved, were selected. In addition, for gray literature, sources were chosen based on their credibility, ensuring that they originated from reputable institutions, industry leaders, or recognized experts within the field of software engineering.

To refine the selection of academic papers, the table 1.1 lists the inclusion and exclusion criteria that we used.

Table 1.1 Inclusion and Exclusion Criteria - Academic Literature

Inclusion criteria	**Exclusion criteria**
Studies must be conducted within the software engineering field.	Papers that focus solely on PoC results, without describing the process or methodology in detail.
Studies must explicitly describe the PoC process or methodology.	Studies that use "proof of concept" interchangeably with prototypes, pilots, or demos without distinguishing the specific PoC process.

The search for academic literature was conducted using the Scopus database, focusing on research published in well-established software engineering journals and conferences. The following query was used to retrieve relevant papers:

```
TITLE-ABS-KEY("proof of concept" OR "PoC" OR "proof-of-concept")
AND ( LIMIT-TO ( SUBJAREA,"COMP" ) )
AND (
   SRCTITLE("ACM/IEEE International Conference on Software Engineering") OR
   SRCTITLE("IEEE Transactions on Software Engineering") OR
   SRCTITLE("Journal of Systems and Software") OR
   SRCTITLE("Information and Software Technology") OR
   SRCTITLE("ACM SIGSOFT International Symposium on Foundations of Software Engineering") OR
   SRCTITLE("Empirical Software Engineering") OR
   SRCTITLE("Proceedings of the ACM on Programming Languages") OR
   SRCTITLE("IEEE/ACM International Conference on Automated Software Engineering") OR
   SRCTITLE("ACM SIGPLAN Conference on Programming Language Design and Implementation") OR
   SRCTITLE("IEEE Software") OR
   SRCTITLE("Mining Software Repositories") OR
   SRCTITLE("International Symposium on Software Testing and Analysis") OR
   SRCTITLE("Software and Systems Modeling") OR
   SRCTITLE("Software: Practice and Experience") OR
   SRCTITLE("International Conference on Software Analysis, Evolution, and Reengineering") OR
   SRCTITLE("ACM Transactions on Software Engineering and Methodology") OR
   SRCTITLE("IEEE International Conference on Software Maintenance and Evolution") OR
   SRCTITLE("International Conference on Tools and Algorithms for the Construction and Analysis of Systems") OR
   SRCTITLE("ACM SIGPLAN Symposium on Principles \& Practice of Parallel Programming") OR
   SRCTITLE("IEEE Annual Computer Software and Applications Conference")
)
```

The search identified 172 documents. After filtering for studies that included "proof of concept" or "PoC" in the title or abstract, 144 studies remained. Closer examination revealed that all of these studies focused on reporting PoC results without detailing the PoC process. Studies that lacked a detailed description of the PoC process were excluded from the analysis. Additionally, many of the studies confused "proof of concept" with "prototype," "minimum viable product (MVP)," and "demo," conflating these distinct concepts.

The analysis of academic literature reveals a notable gap in the literature regarding detailed descriptions of the PoC process in software engineering. Despite the prevalence of PoCs in practice, academic research appears to prioritize reporting on outcomes rather than documenting the methodological steps involved.

The following section present a comprehensive analysis of grey literature. We anticipate that this analysis yield valuable insights into the practical implementation of PoCs in software engineering, filling the gap identified in academic research.

1.2.2 Analysis of grey literature

To address the gap identified in the academic literature regarding detailed descriptions of the PoC process, a systematic analysis of grey literature was conducted. This approach, as proposed by Garousi, Felderer & Mäntylä (2019), allows for the inclusion of diverse sources not traditionally found in academic databases, such as technical reports, white papers, industry blogs, and conference presentations. This broader scope is particularly valuable when investigating emerging practices or topics where academic research might be limited.

1.2.2.1 Quality Assessment and Selection Criteria

A rigorous assessment process was employed to evaluate the quality and relevance of the grey literature, ensuring that only credible sources were included. Following the methodology proposed by Garousi *et al.* (2019), the evaluation was based on the authority and expertise of the source. Specifically, the reputation of the publishing organization or individual author was scrutinized. Publications from well-established institutions or from authors with demonstrable experience in software engineering were considered more reliable. Furthermore, the publication history of the authors or organizations was reviewed to determine their contributions to discussions on PoC processes. This helped to ensure that the sources selected had a consistent record of producing work relevant to the domain.

In addition to evaluating authority, the expertise of the authors was critically assessed. Authors who demonstrated practical experience and technical knowledge in software engineering were prioritized. This was evident through their professional credentials, job titles, and the technical depth of their publications. This assessment aligns with the recommendations from Garousi *et al.*

(2019), which emphasize the need for credible and technically sound sources when including grey literature in research.

The selection of grey literature was guided by specific inclusion and exclusion criteria to ensure relevance to the research focus. Publications were included if they directly related to software engineering and provided a comprehensive explanation of the PoC process or methodology. This ensured the analysis focused on the practical aspects of PoC development, which are often underrepresented in academic research. Publications that only reported PoC outcomes without describing the development process were excluded. This approach kept the focus on understanding how PoCs are developed, rather than on their results. By applying these criteria, the grey literature provided practical insights into PoC implementation, addressing gaps in academic literature, as suggested by Garousi *et al.* (2019).

To refine the selection of grey literature, the the table 1.2 lists the inclusion and exclusion criteria that we used.

Table 1.2 Inclusion and Exclusion Criteria - Grey literature

Inclusion Criteria	**Exclusion Criteria**
Publications must explicitly focus on the software engineering domain.	Publications that solely present PoC results without delving into the process itself were excluded. This criterion ensures that the analysis focuses on the practical "how-to" of PoC development rather than merely its outcomes.
Publications must provide a detailed description or explanation of the PoC process or methodology.	

By employing this dual approach of quality assessment and carefully defined selection criteria, the study was able to incorporate grey literature that was both authoritative and relevant. This integration of diverse sources provided a more comprehensive view of the PoC process.

1.2.2.2 Search Strategy and Stopping Criteria

A systematic search was conducted using Google as the primary search engine to gather relevant grey literature on the Proof of Concept (PoC) process in software engineering. "successful PoC""software engineering,""successful proof of concept""software engineering." Utilizing Google as the primary search engine, a systematic search was performed using the following strings:

```
"how to PoC" AND "software engineering"
"how to proof of concept" AND "software engineering"
"successful PoC" AND "software engineering"
"successful proof of concept" AND "software engineering
```

These search strings were carefully designed to capture a broad range of documents while keeping the focus on practical aspects of PoC implementation within the software engineering domain.

In line with the recommendations Garousi *et al.* (2019), the stopping criteria were established to ensure an efficient and comprehensive review of the grey literature. The search process followed the principle of **theoretical saturation**, meaning it was terminated when no new concepts or insights regarding the PoC process emerged from the reviewed documents. This approach ensured that the search yielded a sufficient understanding of the current practices and methodologies surrounding PoC in software engineering.

Effort was also bounded by including only the top **20 search results for each query** including the inclusion criterias. This limit was considered sufficient to provide a representative sample of the grey literature available on the subject. Furthermore, the review process adhered to the principle of **evidence exhaustion**, extracting all relevant information on PoC practices from the included sources. This strategy allowed for a thorough exploration of the grey literature, contributing to a well-rounded understanding of the PoC process beyond what is typically covered in academic research.

The insights gathered through this systematic search provided the foundation for identifying key characteristics of successful PoC implementations, as analyzed in the next section.

1.3 Identifying PoC characteristics

Building on the findings from the systematic search, this section analyzes the characteristics of PoCs to explore their common elements. The table 1.3 presents the PoC characteristics founded, while table 1.4 provides detailed descriptions of each characteristic and explains their significance in the context of a PoC experiment. Figure 2.1 illustrates the works analyzed for this study.

Table 1.3 PoC Common characteristics

Characteristics	**Reference**
Gain confidence / validate design	Fairbanks (2010); Morris (2016); Burns (2018); Shrivastava *et al.* (2022); Baptista & Abbruzzese (2024)
Quicker experimentation	Burns (2018); Enríquez & Salazar (2018); Beningo (2022); Tune & Perrin (2024)
Solve learning needs	Morris (2016); Ingeno (2018); Burns (2018); Shrivastava *et al.* (2022); Tune & Perrin (2024)
Evaluate risks / discover unknowns / identify gaps	Ingeno (2018); Burns (2018); Shrivastava *et al.* (2022); Tune & Perrin (2024)
Determine feasibility	Helgeson (2009); Shrivastava *et al.* (2022); Tune & Perrin (2024)
PoC should not be a deliverable	Ford *et al.* (2017); Burns (2018); Hinchey (2018); Shrivastava *et al.* (2022); Beningo (2022)
Helps to choose/evaluate a proven technology	Morris (2016); Ford *et al.* (2017); Burns (2018); Shrivastava *et al.* (2022); Brisals & Hedger (2024)
Analyse technical limitation / Performance Needs	Burns (2018); Ingeno (2018)

Table 1.4 PoC characteristics description

Characteristics	**Description**	**Importance for PoC**
Gain Confidence / Validate Design	Ensure the design meets the requirements and expectations.	Critical for the success and viability of the PoC.
Quicker Experimentation	Rapidly test and iterate to refine the solution.	Accelerates the development process and innovation.
Solve Learning Needs	Address the knowledge gaps within the team.	Facilitates effective implementation and reduces the learning curve.
Evaluate risks / discover unknowns / identify gaps	Identify and address any uncertainties or challenges.	Prepares the team for potential obstacles and ensures stability.
PoC should not be a deliverable	The PoC source code must not be used in production systems.	PoC code focuses on hypothesis validation and can be valuable despite not meeting production standards.
Determine Feasibility	Assess whether the PoC is technically and financially feasible.	Avoids pursuing unviable solutions and wasting resources.
Helps to Choose/Evaluate a Proven Technology	Use the PoC to assess the suitability of established technologies.	Minimizes the risk of adopting new, unproven solutions.
Analyse technical limitation / Performance Needs	Investigate any technical constraints that may impact the solution.	Prevents commitment to a technology without full understanding and ensures that performance considerations will be met.

Eight important characteristics emerged as critical for the PoC creation process. Common elements across different PoCs include **gaining confidence or validating designs, evaluating risks, discovering unknowns, identifying gaps**, and **analyzing technical limitations**. These characteristics align with the purpose of PoC experiments, which is primarily risk mitigation.

Quicker experimentation and **PoC should not be a deliverable** was a a characteristics shared in different PoCs, this idea of a PoC is short-live term and ephemeral and reinforces the idea that the PoC code don't be used in production system. Solve **learning needs** is a important

characteristic mentioned in different PoC executions, because many authors see in a PoC a important tools to supports learning. Also, the characteristics **determine feasibility and assess technical limitations** contribute to the decision-making process by providing insights into the feasibility of potential solutions and helping to evaluate technical risks.

These identified characteristics align with the PoC definition reinforce the idea that a PoC is a short-term experiment, which, by nature, has a limited scope. It focuses on assessing feasibility, mitigating risks, and supporting learning and decision-making. In summary, this section provides an analysis of PoC characteristics, which is valuable for practitioners to review and apply in their own PoC experiments.

1.4 Defining PoC in the Software Engineering Domain

As discussed previously, the concept of Proof of Concept (PoC) originated in engineering fields outside software development and was later adapted for software engineering. In the software engineering domain, the definition of Proof of Concept (PoC) often remains vague, with organizations interpreting it differently based on their project-specific needs.

Due to the lack of PoC definition in software engineering, some authors argue that PoC, Prototype and MVP are synonymous, using these terms interchangeably Larman (2002); Fairbanks (2010); Mistrik (2012); Van Hecke (2013); Anderson (2023). In contrast Ingeno (2018); Shrivastava *et al.* (2022); Prateek (2024); TechBohdana (2024), highlight a distinction between the then, emphasizing differences in scope and purpose.

An additional debate concerns whether the code developed in a PoC should be reused in future software versions. While Van Hecke (2013) advocates for code reuse, suggesting it can accelerate development, Ford *et al.* (2017); Burns (2018); Hinchey (2018); Beningo (2022) argue that it should be discarded as its primary purpose is to test specific hypotheses within a limited context, rather than laying the foundation for future development.

Based on these perspectives, we propose a more refined definition of PoC within the context of software engineering. This new definition emphasizes the limited scope of a PoC and its inherent nature as an experimental tool. The proposed definition is as follows:

> **Proof of Concept (PoC) in Software Engineering is a short-term experiment that validates hypotheses within a limited scope by assessing feasibility, mitigating risks, supporting learning and decision-making.**

After identifying the PoC characteristics founded during the analyzes, it becomes possible to link them with the PoC definition mentioned above, the founded characteristics align with the PoC definition. Figure 1.2 illustrates the relationship between the PoC characteristics and the established PoC definition.

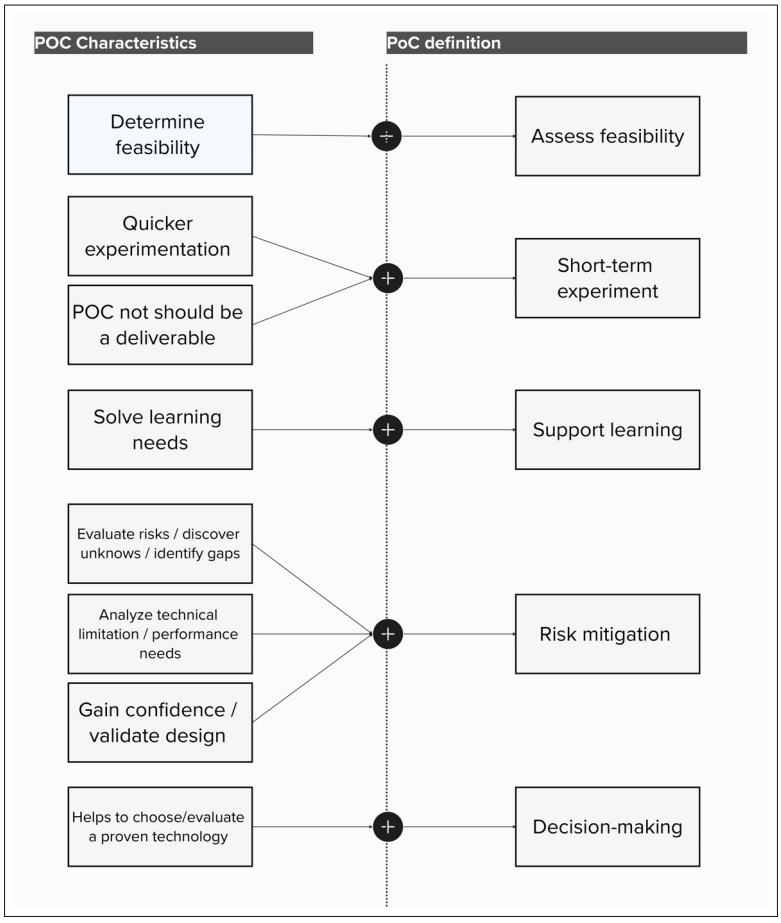


Figure 1.2 PoC Characteristics and their link with a definition

The proposed definition sheds new light on the concept of a Proof of Concept (PoC) by emphasizing its transient and experimental nature. This sets PoC apart from other techniques like Prototypes and Minimum Viable Products (MVPs), which are not inherently ephemeral. Unlike these approaches, PoC focuses on short-term experimentation designed to validate assumptions within a constrained scope, rather than contributing to the development of a production-ready system. As a result, the code produced during a PoC is considered disposable, serving only to validate specific assumptions.

1.5 Conclusion

This chapter reveals the PoC evolution from other fields to software engineering without a clear definition in the software engineering domain. This has caused misunderstandings about what should be considered a PoC in software engineering, leading to disagreements among different authors.

Based on the characteristics of PoC, it was possible to develop a more refined definition of the PoC concept in software engineering. This definition helps avoid confusion between PoC, MVP (Minimum Viable Product), and prototype by clearly delimiting the scope of a PoC. It emphasizes the role of PoCs in supporting decision-making, risk mitigation, feasibility, and learning. As a result, PoC practitioners have a clearer understanding of the objectives of a PoC and characteristics.

CHAPTER 2

POC PROCESS AND FRAMEWORK DEVELOPMENT

This chapter examines the essential characteristics of Proof of Concept (PoC) in software engineering, offering a comprehensive view of the process and the roles involved. Analyzing these characteristics offers insights into how different projects share similar elements, which supports a more consistent and structured approach to PoC execution. These shared elements provide a foundation for understanding how PoCs can be adapted to various contexts while maintaining core principles.

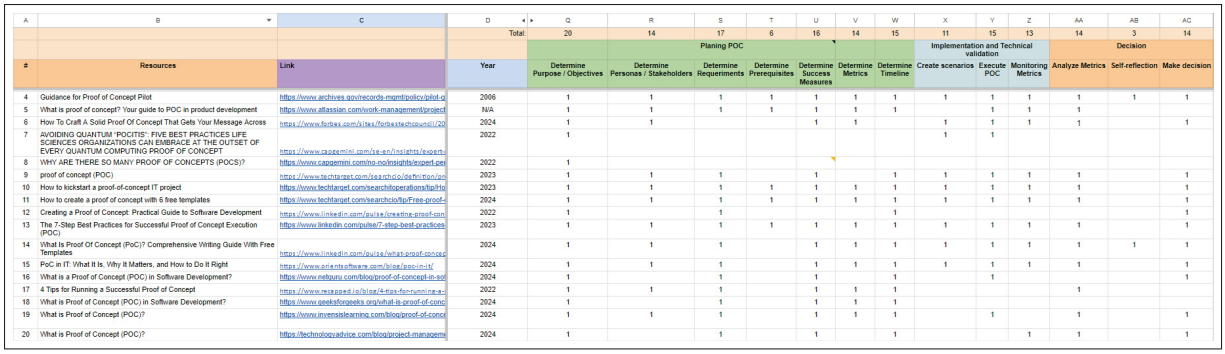
The chapter then outlines the stages of the PoC process, using a Business Process Model and Notation (BPMN) diagram to visually represent each step. This visual aid clarifies the sequence of actions and the decision points involved, allowing practitioners to better grasp the flow of the process. To conclude, the chapter introduces a structured framework that integrates the previously identified characteristics and process steps. This framework serves as a guide for PoC practitioners, enabling them to plan, execute, and evaluate PoCs in a systematic manner.

2.1 Identifying PoC Process

After understanding the characteristics of PoCs in the last section, this section examine the stages involved in the creation process. To identify these steps, an analysis of 20 blog articles was conducted. The selection of these articles followed the guidelines proposed by Garousi *et al.* (2019). Additionally, the systematic analysis of grey literature was guided by a spreadsheet provided by Garousi *et al.* (2019), ensuring a structured approach to the data.

The findings from this analysis are summarized in Figure 2.1 which presents the key data extracted from the reviewed blogs.[1]

[1] https://docs.google.com/spreadsheets/d/1Mh5kbHRuQ_CMGEBvNRnyYgzdmm1aRKyADuXnWI2XZmc



#	Resources	Link	Year	Planing POC: Determine Purpose / Objectives	Determine Personas / Stakeholders	Determine Requeriments	Determine Prerequisites	Determine Success Measures	Determine Metrics	Determine Timeline	Implementation and Technical validation: Create scenarios	Execute POC	Monitoring Metrics	Decision: Analyze Metrics	Self-reflection	Make decision
	Total:			20	14	17	6	16	14	15	11	15	13	14	3	14
4	Guidance for Proof of Concept Pilot	https://www.archives.gov/records-mgmt/policy/pilot-p	2006	1	1	1	1	1	1	1	1	1	1	1	1	1
5	What is proof of concept? Your guide to POC in product development	https://www.atlassian.com/work-management/project	N/A	1	1	1	1	1	1	1		1	1	1		
6	How To Craft A Solid Proof Of Concept That Gets Your Message Across	https://www.forbes.com/sites/forbesbrandvoice/i/20	2024	1	1			1	1		1	1	1	1		1
7	AVOIDING QUANTUM "POCITIS": FIVE BEST PRACTICES LIFE SCIENCES ORGANIZATIONS CAN EMBRACE AT THE OUTSET OF EVERY QUANTUM COMPUTING PROOF OF CONCEPT	https://www.capgemini.com/se-en/insights/expert-	2022	1							1	1				
8	WHY ARE THERE SO MANY PROOF OF CONCEPTS (POCS)?	https://www.capgemini.com/no-no/insights/expert-pe	2022	1												
9	proof of concept (POC)	https://www.techtarget.com/searchcio/definition/pr	2023	1	1	1		1		1	1	1	1	1		1
10	How to kickstart a proof-of-concept IT project	https://www.techtarget.com/searchitoperations/tip/Ho	2023	1	1	1	1	1	1	1	1	1	1	1		1
11	How to create a proof of concept with 6 free templates	https://www.techtarget.com/searchcio/tip/Free-proof-	2024	1	1	1	1	1	1	1	1	1	1	1		1
12	Creating a Proof of Concept: Practical Guide to Software Development	https://www.linkedin.com/pulse/creating-proof-con	2022	1		1				1						1
13	The 7-Step Best Practices for Successful Proof of Concept Execution (POC)	https://www.linkedin.com/pulse/7-step-best-practices	2023	1	1	1	1	1	1	1	1	1	1	1		1
14	What Is Proof Of Concept (PoC)? Comprehensive Writing Guide With Free Templates	https://www.linkedin.com/pulse/what-proof-concep	2024	1	1	1		1	1	1	1	1	1	1	1	1
15	PoC in IT: What It Is, Why It Matters, and How to Do It Right	https://www.orientsoftware.com/blog/poc-in-it/	2024	1	1	1		1	1	1	1	1	1	1		1
16	What is a Proof of Concept (POC) in Software Development?	https://www.netguru.com/blog/proof-of-concept-in-sof	2024	1		1		1		1		1				1
17	4 Tips for Running a Successful Proof of Concept	https://www.recapped.io/blog/4-tips-for-running-a-s	2022	1	1	1		1	1	1				1		
18	What is Proof of Concept (POC) in Software Development?	https://www.geeksforgeeks.org/what-is-proof-of-conc	2024	1		1		1	1	1						
19	What is Proof of Concept (POC)?	https://www.invensislearning.com/blog/proof-of-conc	2024	1	1	1		1	1	1		1		1		1
20	What is Proof of Concept (POC)?	https://technologyadvice.com/blog/project-managem	2024	1		1		1		1			1	1		1

Figure 2.1 Analysis spreadsheet

Three main processes emerged from this analysis, each with distinct steps. The first process in a PoC is the **planning stage**, which consists of seven steps. The primary goal of PoC planning is to determine the factors that contribute to its success. These steps include defining the PoC's purpose or objective, identifying personas and stakeholders, outlining requirements, listing prerequisites, establishing success measures, selecting metrics, and setting a timeline.

The second process involves the **implementation and technical validation** of the PoC. This stage comprises three steps: creating scenarios, executing the PoC, and monitoring metrics. This phase is critical as it generates data that informs and validates the hypotheses made during the planning stage.

Finally, the **decision-making** process consists of three steps: analyzing the metrics gathered during the implementation phase, comparing these results with the expectations set during planning, and making an informed decision based on the outcomes.

For a comprehensive understanding of the PoC process, Figure 2.2 illustrates the complete PoC workflow.

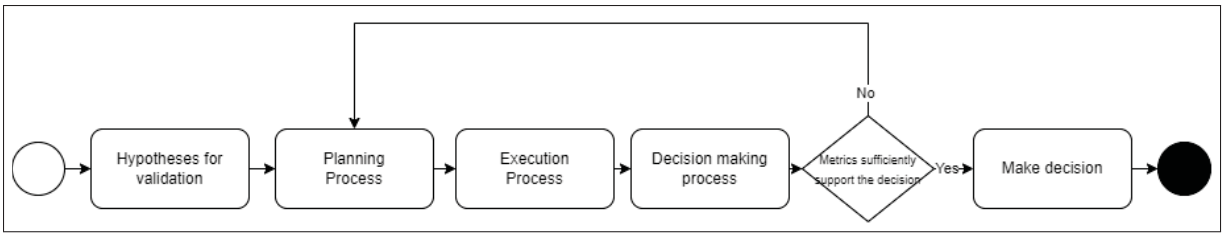


Figure 2.2 PoC Process Diagram

A detailed BPMN diagram, shown in Figure 2.3, provides a comprehensive view of the PoC process. This diagram serves as a valuable tool for PoC practitioners, offering a clearer understanding of the entire PoC process in detail.

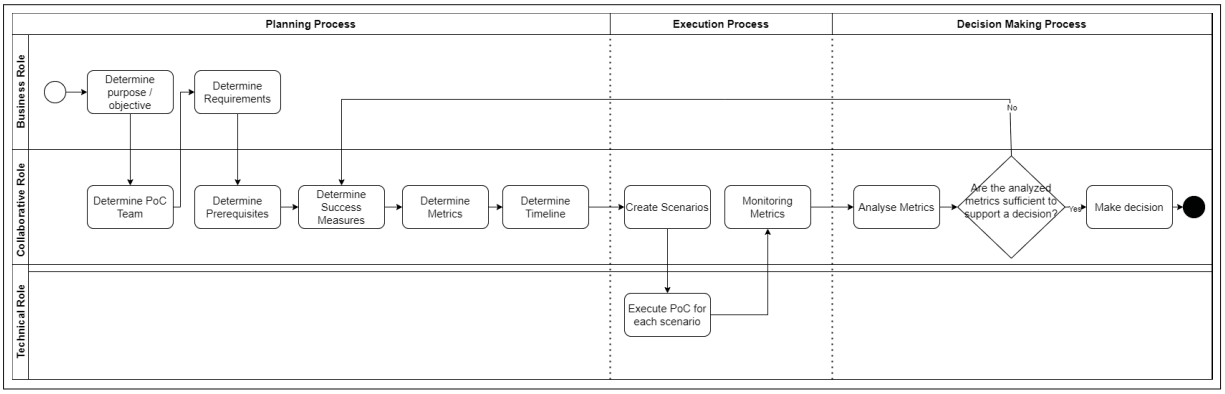


Figure 2.3 BPMN PoC

The next subsection provides a detailed explanation of each step in the PoC process. Understanding these steps in detail is crucial for PoC practitioners, as it offers insight into the fundamental structure of the PoC process and clarifies the role of each step within the overall workflow.

2.1.1 Detailed Steps of a PoC Process

The three main processes of a PoC were identified in the previous section. For PoC practitioners, it is essential to understand each step of the PoC process in detail. This subsection provides a deeper analysis of the steps involved in a PoC process.

The **planning phase** serves as the foundation for the success of a PoC. This phase involves defining key elements that guide both the execution and evaluation of the PoC. The following steps are integral to the planning phase:

- **Determine Purpose/Objectives:** This involves clearly defining the purpose of the PoC, whether it is solving a specific problem, testing a hypothesis, or exploring an innovation. Establishing well-defined objectives is crucial, as they provide clear direction and serve as a guide for the entire experiment.
- **Determine Personas/Stakeholders:** Focuses on identifying the individuals or groups who will be directly or indirectly impacted by the PoC. This includes end-users, project sponsors, technical teams, and other relevant parties. Understanding the needs, expectations, and concerns of these stakeholders ensures that the PoC remains relevant and addresses the correct problems. For more details on PoC team members based on analyzed references, see Appendix A, Table III-1.
- **Determine Requirements:** Involves specifying the features, functionalities, and conditions that the product, system, or service must fulfill. These requirements are typically derived from stakeholder needs and serve as guidelines for validating the initial assumptions. Requirements cover a broad range of aspects, including technical, functional, and non-functional needs.
- **Determine Prerequisites:** Before the PoC can begin, certain conditions or resources must be in place. Examples: o Technical infrastructure (hardware, software, network): The server infrastructure must be upgraded before the PoC can start. o Data availability and accessibility: We have all the necessary access and permissions to execute a PoC? o Skilled personnel: The team needs to complete training on the new technology before working on the PoC. o Budgetary approvals: A budget of X must be approved before initiating the PoC.
- **Determine Success Measures:** How will you know if the PoC is successful? This involves establishing clear, measurable criteria for evaluating the PoC's outcomes. Success measures could include: o Quantitative metrics (e.g., performance benchmarks, cost savings) o Qualitative assessments (e.g., user feedback, stakeholder satisfaction).

- **Determine Metrics:** While success measures define the overarching criteria for success, metrics provide specific, quantifiable data points for tracking progress and evaluating results. Examples include: o System response time o User error rate o Cost per transaction.
- **Determine Timeline:** Establishing a realistic timeline is essential for managing expectations and ensuring the PoC stays on track. The timeline should include: o Key milestones (e.g., completion of development, testing phases) o Deadlines o Resource allocation.

The process **Implementation and Technical Validation** has three step, this phase marks the transition from theoretical planning to practical execution. It involves putting the PoC into action and rigorously testing its capabilities to ensure it meets the desired objectives:

- **Create Scenarios:** This involves designing specific scenarios or use cases that closely simulate real-world situations. These scenarios should be tailored to test the core functionalities and features of the PoC under different conditions. Purpose: The purpose of creating scenarios is to gain a comprehensive understanding of how the PoC performs in various situations. It helps identify potential issues, bottlenecks, or areas for improvement before full-scale implementation.
- **Execute PoC:** This is the hands-on implementation of the PoC, where the proposed solution or technology is deployed in a controlled environment. It involves setting up the necessary infrastructure, configuring the system, and running the predefined scenarios. Purpose: The goal of executing the PoC is to gather real-world data and observe how the solution performs in practice. This provides valuable insights into its feasibility, effectiveness, and potential impact.
- **Monitoring Metrics:** This involves continuously tracking and measuring key performance indicators (KPIs) throughout the PoC's execution. These metrics could be related to system performance, user experience, resource utilization, or any other relevant aspect of the PoC.Purpose: Monitoring metrics provides valuable feedback in real time, allowing for quick identification of issues and adjustments to the PoC as needed. It also helps in collecting data for later analysis and decision-making.

The **process decision** is the culmination of the PoC process, where the insights and data gathered are synthesized to make informed choices about the project's future:

- **Analyze Metrics:** This involves a comprehensive analysis of the data collected during the PoC's execution. It includes reviewing the performance metrics, comparing them against the predefined success criteria, and identifying any trends or patterns. Purpose: The purpose of analyzing metrics is to gain a deep understanding of the PoC's strengths and weaknesses, evaluate its overall effectiveness, and determine if it met the initial objectives.
- **Make Decision:** Based on the analysis of metrics and the insights gained from self-reflection, a decision is made regarding the future of the project. This decision could be to proceed with full-scale implementation, modify the approach, or discontinue the project altogether. Purpose: The purpose of this step is to translate the findings of the PoC into actionable next steps. It ensures that resources are allocated efficiently and that the organization makes informed decisions based on evidence.

For the PoC practitioners is important to understand how each step in a PoC process is made, how the decision-making process works and how important is each step of the PoC process. This subsection provided detailed information about each steps of a PoC process. This is important to understand the backbone of a PoC Process.

2.2 PoC Framework

After a thorough analysis and deep understanding of both the PoC characteristics and the PoC process, this section presents a framework for PoC execution. The framework was developed to guide PoC practitioners through each stage of the process, ensuring a structured and systematic approach. An version of the framework is available in an online document[2].

The framework is composed of four main sections, which emerged from the in-depth analysis of PoC characteristics and processes. These sections provide a clear roadmap for planning,

[2] https://docs.google.com/spreadsheets/d/1Mh5kbHRuQ_CMGEBvNRnyYgzdmm1aRKyADuXnWI2XZmc

executing, and evaluating PoCs, addressing the critical elements necessary for a PoC experiment. The key sections of the PoC Framework are as follows:

1. General information;
2. Requirements, prerequisites, success measures and metrics definition;
3. Timeline and PoC conclusion;
4. Scenario description.

The following subsections provide a detailed explanation of each of the main sections of the framework. Each subsection describes how the primary categories were developed, offering PoC practitioners a deeper understanding of the rationale behind the framework. This detailed breakdown enables practitioners to effectively apply the proposed framework in their own PoC processes.

2.2.1 General information

The section "General information" is important to present the PoC purpose in a concise and systematized manner, this section is composed by **three specifics areas**.

In the **"Determine Purpose/Objective"** area, the rationale for the creation of the PoC should be documented. This should be directly linked to the initial hypotheses that led to the PoC's development.

"Team" area helps the PoC practitioner identify which teams are responsible for executing the PoC. The framework presents some teams based on insights from the literature to facilitate the process. While certain technical and business teams are suggested, it is possible that other teams not yet identified may also be involved in your PoC. The suggested teams are based on the information presented in Table III-1. In the **"Characteristics"** area, the characteristics described in the table provide guidelines for PoC creation 1.4. This table was created based on various PoCs, increasing the likelihood that your PoC will share similar characteristics.

The figure 2.4 illustrates the general section of the framework.

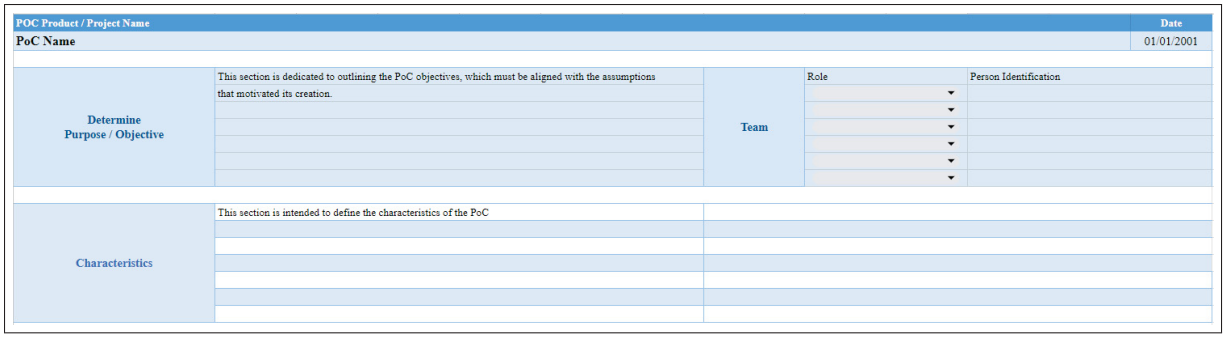


Figure 2.4 General information's

2.2.2 Requirements, prerequisites, success measures and metrics definition

This section is important to link the requirements, success measures and metrics, in a systematized way. This section is also important to verify all the prerequisites needed to execute the PoC. This section is composed by **four specifics areas**.

The **Requirements** area is intended for capturing the requirements identified by stakeholders. Following this, the section addresses the **prerequisites** necessary for executing the PoC. The requirements are directly tied to the **success measures**, and in turn, the success measures are closely linked to the **metrics**.

This section serves as the point where the requirements defined by stakeholders can be connected with the success measures and metrics defined by the PoC team, while also taking into account the prerequisites required for PoC execution.

The figure 2.5 illustrates the section dedicated to requirements, prerequisites, success measures, and metrics.

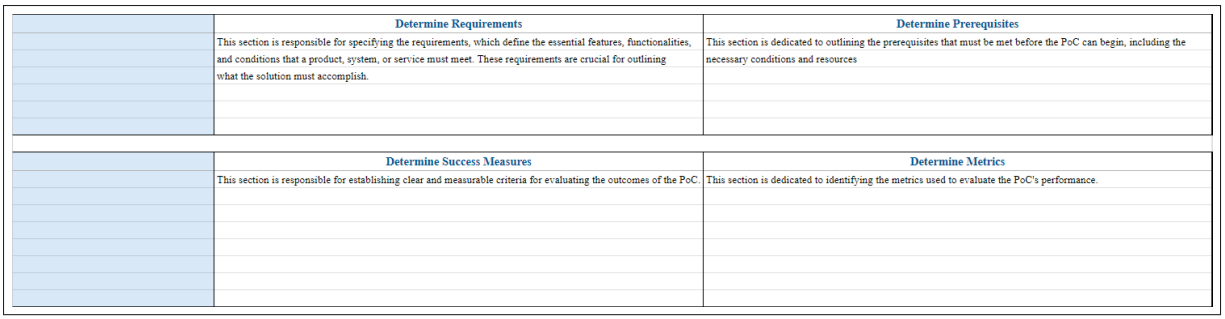


Figure 2.5 Requirements, Prerequisites, Success measures and metrics

2.2.3 Timeline and PoC conclusion

The Timeline and PoC Conclusion section, illustrated in Figure 2.6 is designed to outline the timeline of the PoC. Due to the many unknowns involved in executing a PoC, it is recommended to structure the timeline around logical phases of PoC development rather than fixed delivery dates.

This section also includes the overall conclusion of the PoC. After executing the test scenarios, stakeholders will use this space to determine whether the initial hypotheses were confirmed or disprove. This specific area of the framework will assist decision-makers in gaining a comprehensive view of the experiment's success.

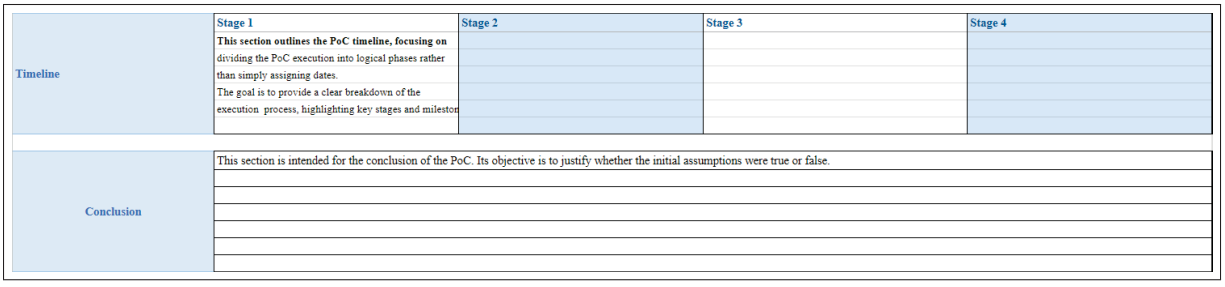


Figure 2.6 Timeline and PoC conclusion

2.2.4 Scenario description

The figure 2.7 illustrates the execution scenarios of the PoC. A scenario may be a single instance or involve multiple scenarios.

This section includes a specific area for scenario description, where the testing objectives for each scenario should be outlined. The scenario is also directly tied to the PoC's requirements, which is why a dedicated section for requirements is included here. The requirements are closely linked to success measures and metrics, and these PoC characteristics are grouped within the scenario to enhance comprehension.

Finally, the section concludes with an evaluation of the scenario. This part serves to summarize the conclusions drawn from the scenario. It is possible that new metrics or success measures may emerge during the scenario evaluation, which should feed back into the initial planning phase of the PoC.

Scenario Name				
Description	SCENARIO DESCRIPTION			
Scenario Measures	**Requirements**	**Success Measures**	**Metrics**	**Metric Values**
	Identify which requirement is directly linked to the success measures	Identify which metrics are linked to success measures	Metrics that should be analyzed during scenario execution	Value collected from metrics
Evaluation	This section is responsible for evaluating a proposed scenario, with the objective of identifying new insights related to its execution. As a result, new metrics may be developed to refine and enhance the PoC process.			

Figure 2.7 Scenario description

2.2.5 Conclusion

This chapter introduces a framework to guide Proof of Concept (PoC) processes in software engineering, detailing the stages of planning, execution, and evaluation. By analyzing PoC characteristics and systematically organizing the PoC process, the framework aims to provide clarity and consistency to a PoC process by defining clear objectives, assessing feasibility, managing risks, and supporting decision-making.

Incorporating grey literature alongside academic sources adds practical relevance, bridging the gap between theory and industry practice. This dual-source approach also addresses academic research gaps, particularly in detailing the PoC process and its essential characteristics. The resulting framework emphasizes PoCs as structured, short-term experiments for hypothesis validation and serves as a flexible guide adaptable to various project contexts, enhancing the reliability and applicability of PoC findings.

CHAPTER 3

FRAMEWORK EVALUATION: A CASE STUDY OF DATABASE VERSIONING IN THE BANKING DOMAIN

3.1 Introduction

Database versioning management is a critical requirement for financial institutions, driven by operational needs and strict regulatory demands. In regulated financial environments, ensuring the consistency and traceability of database changes is essential for maintaining compliance with financial standards.

This chapter evaluates the effectiveness of the proposed PoC framework through a real-world case study, focused on database versioning within a Java-based banking application. The case study leverages documentation from the bank's internal Confluence pages, providing a detailed basis for evaluating two widely-used database versioning tools, Flyway and Liquibase. The aim is to assess which tool better aligns with the bank's technical and compliance needs, while also examining how the PoC framework supports decision-making and regulatory adherence.

By focusing on both technical and business requirements, this chapter evaluates how the PoC framework provides a repeatable and objective process for making data-driven decisions, particularly in choosing the most suitable tool for the company's needs. This chapter will demonstrate how the framework helped make the right choice between Flyway and Liquibase by analyzing requirements, success metrics, evaluation criteria, and the specific characteristics of the PoC process.

By analyzing the PoC framework application in this context, this chapter reveals its broader applicability for similar decision-making scenarios. The implications of these findings extend to future implementations, particularly in highly regulated industries where ensuring compliance is critical to business operations.

3.2 Applying the PoC Framework

The PoC framework, developed to address both technical and business concerns, was applied to this case study using a structured approach. The framework involves a multi-step process that includes stakeholder identification, requirements gathering, scenario planning, and performance evaluation. By utilizing Confluence documentation as the primary reference, the framework ensured that stakeholder needs and compliance requirements were fully met while remaining aligned with the bank's technical environment.

Appendix II includes includes figures showing the results of the framework execution. Figure II-1, detailing the framework session covering objectives, team roles, characteristics, requirements, and prerequisites, Figure II-2 illustrating the session on success measures, metrics, and timeline, Figure II-3 and II-4 presenting the PoC conclusion and description of tests scenario. For a complete analysis, refer to the detailed document available online[1].

In the following sections, we will explore each phase of the framework application, providing detailed insights into how the framework was implemented in the case study.

3.2.1 Planning

In the planning phase, internal records were directly consulted, offering detailed information on previous database migrations, success metrics, and key pain points from past projects. The PoC framework utilized this historical data to address both technical and operational challenges identified in earlier initiatives. The following subsections examine how the framework enhanced Purpose/Objectives, Personas/Stakeholders, Requirements, Prerequisites, Success Measures, Metrics, and Timeline.

[1] https://docs.google.com/spreadsheets/d/1o0T8DBjjQbyAQgNdqhHnRSUwiKKMpOt2LQDejM_s248/edit?usp=sharing

3.2.1.1 Purpose/Objectives

The PoC objectives emphasized the need for robust audit trails, reliable rollback capabilities, and alignment with regulatory standards. Two widely-used database versioning tools, Flyway and Liquibase, were evaluated for their ability to handle schema versioning, rollback mechanisms, and compliance with banking industry regulations. Each tool was evaluated based on the following aspects:

1. Comprehensive schema versioning with auditability.
2. Seamless integration with existing CI/CD pipelines as per documented automation workflows.
3. Compliance with strict regulatory requirements detailed in Confluence.

The objective of this PoC was documented by stakeholders on the PoC framework and shared in a Confluence page, this approach provided a clear and accessible record of the objective. Documenting the objective in Confluence ensured clarity and accessibility for all stakeholders, with automatic version history capturing any changes and enabling a traceable evolution of the objective. By clearly defining these objectives in the framework, the PoC remained aligned with both the technical and business requirements.

3.2.1.2 Defining Personas/Stakeholders

Identifying relevant stakeholders is important to ensure that the PoC addresses diverse perspectives from various viewpoints, contributing to a more comprehensive understanding of the problem. Historical data provided important insights of previous database versioning efforts, which were instrumental in defining the following key stakeholders:

1. **Database Administrators (DBAs)**: As detailed in previous Confluence migration projects, the DBAs were focused on tracking schema changes and ensuring safe rollbacks in case of errors.
2. **Compliance Officers**: According to compliance guidelines on Confluence, they required detailed audit logs for every database operation to ensure alignment with financial regulations.

3. **Developers**: Their priorities, as documented in the CI/CD guidelines, were to ensure that the tools could integrate smoothly with the bank's development workflows and automate migrations without affecting system uptime.
4. **IT Managers**: Based on system performance benchmarks available in Confluence, IT managers sought tools that could scale with the bank's growth while maintaining system performance.

3.2.1.3 Requirements

The Confluence documentation provided a well-defined list of both functional and non-functional requirements gathered from previous database versioning projects. These requirements were integral to the success of the PoC:

1. **Functional Requirements**: Version control for both schema and data changes, rollback mechanisms, and compatibility with Oracle and PostgreSQL databases, as per the bank's tech stack guidelines.
2. **Non-functional Requirements**: Minimizing downtime during migrations, performance under high transaction volumes, and integration with the CI/CD pipeline, all of which were thoroughly documented in Confluence.

3.2.1.4 Prerequisites

To execute this PoC, the software developer must have access to a test environment setup that follows established procedures from internal records, ensuring that all test scenarios mirror real-world use cases. This involves **setting up isolated database instances** in line with Confluence-based technical configurations, preparing migration scripts based on documented business logic and previous migration efforts, and **configuring the CI/CD pipeline** according to standardized procedures outlined on Confluence. Additionally, developers must have **network and database access**, **administrator privileges on their workstations**, and a valid Docker **license** to facilitate the setup and testing process.

3.2.1.5 Success Measures

The success criteria align directly with the bank's internal Confluence system, drawing from historical performance metrics to ensure consistency. First, the **time required to apply migrations** is evaluated, referencing documented performance reports from past migrations. Additionally, the **audit logs are assessed for completeness and clarity**, prioritizing compliance in line with internal documentation. Lastly, the **effectiveness of rollback procedures** is scrutinized, guided by previously documented rollback scenarios to ensure reliability.

3.2.1.6 Metrics

Metrics draw from established success measures documented in prior PoC evaluations within internal Confluence records. **Migration time** is tracked in seconds or minutes, using benchmarks from past performances as references. **Audit log quality** is assessed based on compliance standards outlined in Confluence documentation. **Rollback procedures** are evaluated for efficiency, focusing on the time needed to revert to a prior state, in alignment with documented guidelines. **System resource usage**, including CPU and memory consumption, follows metrics from previous resource utilization data. Additionally, the **ease of integration with Java** applications is considered to gauge overall usability.

3.2.1.7 Timeline

In the **first stag**e, the environment setup and preparation of migration scripts will be conducted. This phase is essential to ensure that the development environment is ready and that all previously identified prerequisites are met. During this stage, evaluation scripts will be created based on the defined test scenarios.

In the **second stage**, the test scenarios will be executed using the Flyway tool. Previously established metrics will be measured and recorded, along with the results obtained from Flyway.

The **third stage** involves executing the same test scenarios, this time with the Liquibase tool. Metrics will again be measured and recorded, allowing for a direct comparison with the results from Flyway.

Finally, in the **last stage**, an analysis of the collected metrics will be carried out, enabling developers to assess which solution integrates best with the existing Java development environment, based on the experience gained with both tools.

3.2.2 Implementation and Technical Validation

The implementation phase relied heavily on previously documented migration and validation processes available within Confluence. Every step was aligned with historical practices, ensuring that the PoC accurately reflected the bank's operational and compliance needs.

3.2.2.1 Scenario Creation

The test scenario was created with consideration for the requirements defined by the team, as well as the established success metrics. The scenario was designed to ensure that all metrics can be thoroughly evaluated, covering various aspects of database migrations and transformations relevant to the bank's operations.

This scenario evaluates capabilities across a comprehensive set of migration tasks, designed to reflect real-world scenario. It includes the creation and modification of tables **(DDL scripts)** to evaluate handling of structural changes, as well as CRUD operations **(DML scripts)** that represent the bank's common data manipulation tasks. It simulates **parallel development** by multiple developers to identify potential conflicts and assesses **rollback and version control** functionalities, referencing prior rollback tests.

The scenario also covers **complex schema changes with dependencie**s, reflecting challenging migrations from past projects. **Data migration** and transformation processes are included to examine each tool's ability to handle typical migration needs. Additionally, it simulates

production migration with an emphasis on **minimizing downtime**, ensuring tools can perform efficiently in live environments. Finally, **auditing and compliance** checks are incorporated to confirm **alignment** with documented **audit and regulatory standards**, verifying that both tools support the bank's compliance requirements.

3.2.2.2 Execution of the PoC

The execution of the PoC was carried out in a development environment using a predefined database test script. This script performed a variety of operations, including both Data Definition Language (DDL) and Data Manipulation Language (DML) tasks. A total of eight scripts were executed, each focusing on different aspects of database versioning, with a final evaluation conducted to test rollback functionality after the execution of the eighth script.

Both Flyway and Liquibase were tested using these scripts to assess their ability to meet the functional and non-functional requirements identified in the earlier phases. The functional requirements included version control for both schema and data changes, rollback mechanisms, and compatibility with Oracle and PostgreSQL databases. Non-functional requirements such as minimizing downtime during migrations, performance under high transaction volumes, and seamless integration with the CI/CD pipeline were also rigorously tested.

All relevant metrics were recorded for future auditing and traceability. This documentation serves as a comprehensive resource for evaluating the long-term effectiveness of both tools within the bank's operational framework. Detailed logs for each test scenario were also maintained, providing a robust dataset for comparing Flyway and Liquibase against the predefined requirements.

3.2.2.3 Monitoring Metrics

Throughout the Proof of Concept (PoC), key performance metrics were monitored and recorded using Confluence as the primary documentation tool. The monitoring process played a critical role in tracking system performance, compliance adherence, and rollback efficiency, providing

real-time feedback for each scenario. The metrics collected allowed for a direct comparison between Flyway and Liquibase, offering insights into how each tool performed under different conditions. The metrics **time to apply migrations**, **audit log quality**, **rollback time**, **system resource usage**, and **ease of use** were consistently tracked to ensure both tools met the technical and regulatory requirements.

3.2.3 Decision-Making Process

This section synthesizes the insights gathered during the study case implementation, focusing on evaluating two different technologies. The goal is to provide a data-driven recommendation based on the observed results, ensuring that the decision is met the objectives of the company specifications.

3.2.3.1 Analysis of Metrics

Once the scenarios were executed, the results were analyzed by comparing them to documented benchmarks in Confluence. The tools were evaluated based on their alignment with the documented success criteria, which included performance, auditability, and rollback functionality.

While Liquibase demonstrated superior auditability and more robust rollback features, the development team expressed a preference for Flyway due to its simpler integration with SQL syntax. The team prioritized ease of use, particularly because Flyway allowed for writing migrations directly in SQL, which aligned better with the bank's existing development practices, as documented in Confluence.

However, it became evident during the analysis that when the team opted to write database migrations using raw SQL scripts (a feature supported by both tools), the rollback functionality - one of Liquibase's key strengths - was no longer applicable. In scenarios where raw SQL syntax was used, rollback operations would need to be handled manually, thereby reducing the practical advantage Liquibase had in this area.

3.2.3.2 Final Decision

The PoC framework was instrumental in providing a structured and rigorous approach to evaluating Flyway and Liquibase. Through the well-defined scenarios and metrics established in the planning phase, the team was able to assess the performance, ease of integration, and compliance features of both tools under real-world conditions. Each test scenario, from schema migration to rollback capabilities, contributed critical insights that shaped the final decision.

While Liquibase offered superior rollback features, Flyway's simplicity, ease of use, and compatibility with SQL scripts made it the preferred choice. These features better met the team's workflow and reduced the overall complexity of the migration process. While Flyway lacks automatic rollback for raw SQL migrations, it fulfilled the bank's requirements in a more straightforward and efficient manner compared to Liquibase. This decision reflects the bank's operational priorities, where simplicity and integration with existing SQL practices took precedence over advanced rollback features.

The final recommendation was to adopt Flyway, as it provided a faster, more intuitive solution that met the team's needs for database versioning and compliance, even without the automated rollback feature.

3.2.4 Conclusion

The PoC framework, implemented in the banking domain, presents a methodology that is adaptable to industries where regulatory compliance is important. By promoting a structured approach to documentation and validation, the framework improves operational efficiency while ensuring that compliance requirements are met. Its systematic process emphasizes the identification of the roles of the PoC team, the clarification of requirements, and the verification of prerequisites before execution, which reduces risks and supports clear decision-making. This organized pathway ensures that relevant factors are addressed in each PoC phase, enabling organizations to minimize uncertainties and make better informed decisions.

In applying the framework, this study saw significant improvements in the evaluation of PoC scenarios. Clear **definitions of requirements** and thorough management of prerequisites became central to the process, allowing consistent tracking of success measures, relevant metrics, and timeline organization, all of which strengthened PoC reliability. Previously, ambiguities in role distinctions among stakeholders were resolved by **restructuring team responsibilities**, which improved communication between technical and business teams and ensured that all interests were aligned from the beginning. The framework also clarified **functional and non-functional requirements**, an area where previous documentation fell short, thus preventing potential misalignment in PoC objectives.

In addition, the role of the framework in the preparation of technical and organizational requirements , such as tool configurations and access rights, avoided delays and ensured seamless progress across defined phases. Real-time performance monitoring during these phases provided important information, aligning PoC results with project goals. Defining **success measures** and selecting relevant **metrics** contributed to more targeted evaluations, with specific indicators of compliance, business rules, and technical performance, which offered a comprehensive assessment of proposed solutions.

Furthermore, the framework established a well-defined **timeline** for each PoC phase, aligning the parties involved and maintaining efficiency during the process. A range of test scenarios were developed to fully cover technical aspects and business requirements, ensuring that diverse team expectations were met. Finally, the framework's structured **evaluation process** facilitated a collaborative analysis among stakeholders, reinforcing the **decision-making** process with a balanced perspective on each PoC outcome.

Overall, the framework not only helped to better structure the PoC execution process but also provided a solid foundation for informed decision-making, ensuring that the company's needs were addressed.

CHAPTER 4

FRAMEWORK EVALUATION: PERFORMANCE BENCHMARKING JAVA VS. GOLANG

4.1 Introduction

This chapter evaluates the proposed framework by applying it to a publicly available YouTube tutorial that benchmarks the performance of Java (Quarkus) and Golang (Fiber) applications within Kubernetes Putra (2024). The goal is to conduct a reverse engineering analysis to explore how the framework can be effectively applied to real-world scenarios. By utilizing this public study case, this chapter not only tests the framework against a practical example but also provides a replicable scenario that other researchers can use for further analysis.

The main focus of this chapter is to determine whether integrating the proposed framework could enhance the accuracy and effectiveness of the Proof of Concept (PoC) presented in the Putra (2024). By critically analyzing the PoC results, this chapter assesses whether the framework could have added rigor and depth to the original analysis, potentially yielding more precise insights or more efficient outcomes.

The chosen case study was selected based on its clear objective, technical evaluation, and the collection of metrics that support informed decision-making. The study by Putra (2024) was specifically chosen because it meets these criteria. The case study focuses on a well-defined goal: conducting a performance benchmark between Quarkus and Fiber within a Kubernetes environment.

Additionally, to ensure the reliability of the case study, it has been evaluated using the grey literature quality assessment criteria proposed by Garousi *et al.* (2019). A detailed evaluation of the grey literature quality for this case study can be found in Appendix 1, where the authority, purpose, methodological transparency, and relevance of the source are thoroughly assessed.

This chapter concludes with a detailed discussion of the results, focusing on the impact of applying the framework on understanding and executing the PoC, and drawing insights from

this analysis. This reflection serves to assess the utility and flexibility of the framework in various practical scenarios, particularly when applied to PoCs that were not originally structured according to a structured process.

4.2 Applying the PoC Framework

This section focuses on applying the PoC framework to a case study that compares the performance of Java (Quarkus) and Golang (Fiber) on Kubernetes. Through reverse engineering, the PoC presented in the case study is systematically analyzed by segmenting the process into clearly defined stages: Planning, Implementation and Technical Validation, and Decision Making.

Appendix III includes includes figures showing the results of the framework execution. Figure III-1, detailing the framework session covering objectives, team roles, characteristics, requirements, and prerequisites, Figure III-2 illustrating the session on success measures, metrics, and timeline, Figure III-3, III-4 and III-5 presenting the PoC conclusion and description of tests scenario. For a complete document, refer to online documentation [1].

4.2.1 Planning

The case study defines the primary objective of the PoC: to determine which technology, Java (Quarkus) or Golang (Fiber), performs faster within a Kubernetes environment under similar configurations, thereby avoiding inconsistent results due to varying technologies or servers.

The study case does not address important planning stages such as stakeholder identification and detailed requirements specification. By applying the framework, these stages are systematically inferred, filling the gaps left in the original PoC. This approach allows for a structured analysis, where each step of the PoC is discussed in detail, following the framework's guidelines.

[1] https://docs.google.com/spreadsheets/d/1o0T8DBjjQbyAQgNdqhHnRSUwiKKMpOt2LQDejM_s248/edit?usp=sharing

4.2.1.1 Purpose/Objectives

The primary objective of this PoC is to evaluate the performance characteristics of Java (Quarkus) and Golang (Fiber) applications within Kubernetes. Specifically, the PoC aims to determine which technology exhibits faster performance under similar conditions, with particular attention to avoiding technological or server-related inconsistencies, see Apendix III-1.

4.2.1.2 Defining Personas/Stakeholders

Although Putra (2024) does not explicitly identify stakeholders, an analysis of the data in Table III-1 allows for an inference of key participants. This table highlights the specific teams and individuals within the organization who are closely involved with the Proof of Concept (PoC) process.

DevOps Engineers focus on resource utilization and efficiency, especially in cloud environments where costs are closely linked to resource usage. **Cloud Architects** are primarily interested in assessing which technology provides better scalability and performance, informing architectural decisions for future projects. **Developers** concentrate on ease of development and performance optimization, aiming to improve overall application effectiveness, see Apendix III-1 for a framework image.

4.2.1.3 Requirements

Although Putra (2024) does not explicitly outline the key requirements, the application of the PoC framework has enabled a systematic identification and structuring of these requirements. **The functional requirements** include the need for precise monitoring of CPU and memory usage throughout the application's deployment and operation, along with the measurement of latency. Additionally, evaluating startup times and their influence on scalability within the Kubernetes environment is essential.

The **non-functional requirements** focus on scalability, where the application must demonstrate the capability to scale rapidly in response to varying load demands. Furthermore, cost efficiency is a priority, requiring that resource usage be optimized to minimize costs within the cloud environment, see Apendix III-1.

4.2.1.4 Prerequisites

Although Putra (2024) does not clearly define the prerequisites necessary for executing the PoC, the application of the PoC framework has allowed these prerequisites to be systematically identified as follows:

- Access to Prometheus and Grafana for monitoring and logging.
- Access to an S3 bucket in AWS for file storage.
- Read and write access to a PostgreSQL database server.
- Access to a Kubernetes cluster with appropriate configurations.
- A GitHub repository containing the source code for both Java and Golang applications.

4.2.1.5 Success Measures

While the success metrics for the PoC were broadly outlined by Putra (2024), a more detailed analysis through the PoC framework has revealed three new success measures also disponible in the Appendix III-2, that can be analyzed:

- **Latency Performance:** Measured with an emphasis on the system's responsiveness under increasing load conditions.
- **Resource Efficiency:** Monitoring and minimizing CPU and memory usage during both idle and active states, ensuring low resource consumption during non-peak periods while maintaining or improving overall performance.
- **Scalability:** Evaluated by the application's ability to scale efficiently in response to load changes, including the time required to start and become ready after a scale event, which is influenced by factors such as application image size.

4.2.1.6 Metrics

During the tecnology evaluation Putra (2024) utilize several key metrics to evaluate performance, including:

- **Startup Time:** The time taken for the application to become ready in a Kubernetes pod, measured from pod scheduling to readiness.
- **CPU Usage:** CPU usage during idle and load states, with a focus on both the average and peak usage.
- **Memory Usage:** Memory consumption, with a focus on differences between idle and active states, and how these variations impact performance.
- **Latency (p99):** Latency at the 99th percentile, providing insights into the worst-case response times, particularly under high load conditions.
- **Request Handling Capacity:** The number of requests per second (RPS) handled by each application under varying load conditions, indicating the system's ability to scale.

In addition to these, the application of the PoC framework has revealed additional metrics that were not originally considered in the study case:

- **Image Size:** The size of the Docker images used for each technology, which significantly impacts startup time and the overall efficiency of scaling in cloud environments.
- **File Operation Latency:** The time taken for file operations, including reading from the local file system and uploading to S3, as these operations are critical in real-world scenarios.
- **Database Operation Latency:** The time required to write metadata to the database, which is essential for understanding the performance of the application in handling data-intensive tasks.

These additional metrics provide a more comprehensive evaluation of the applications, covering aspects such as efficiency, scalability, and performance under real-world conditions.

4.2.1.7 Timeline

Putra (2024) does not provide a timeline for the PoC execution. However, by applying the PoC framework, it is possible to divide the execution into smaller, logical phases, creating a structured process and foundation for a timeline.

The **first phase** is the setup of prerequisites, involving the configuration of essential elements such as the S3 bucket for storage, access to Prometheus and Grafana for monitoring, and the preparation of the Kubernetes cluster. This setup phase ensures that all foundational resources are in place before performance testing begins.

The **second phase** focuses on baseline performance testing. After establishing the prerequisites, baseline tests are conducted to evaluate the behavior of the Java (Quarkus) and Golang (Fiber) applications in an idle state. These tests measure initial CPU and memory usage, startup times, and other fundamental metrics, forming a baseline for future comparisons under load.

The **third phase** involves load testing, where gradually increasing the load on the applications. This phase captures key metrics like latency (including p99), CPU and memory usage, and request handling capacity (RPS), providing insights into application performance under various demand levels.

The **final phase** is analysis and documentation. In this phase, the collected data is analyzed, comparing the performance of Java (Quarkus) and Golang (Fiber) applications across all metrics. The findings are documented to inform future decisions and optimizations.

This structured breakdown of the PoC execution provides a clear framework that can guide the development of a detailed timeline, allocating focus and resources to each phase. This systematic approach enhances both the project's management and the reliability of PoC outcomes.

4.2.2 Implementation and Technical Validation

The implementation phase followed a structured approach to benchmark the Java and Golang applications under controlled conditions.

4.2.2.1 Scenario Creation

Putra (2024) does not explicitly define test scenarios, but after applying the framework, three distinct scenarios emerges.

The **first scenario**, available in Appendix III-3, focuses on REST API response time, measuring latency for simple REST API calls under varying load conditions. This scenario aims to assess the responsiveness of each application when exposed to different demand levels.

The **second scenario**, available in Appendix III-4, examines file handling and database interaction, evaluating the time required to manage files and interact with an S3 bucket and PostgreSQL database. This setup simulates real-world operations, where both applications must process and store data efficiently.

The **third scenario**, available in Appendix III-5, addresses application startup and scalability, analyzing how quickly each application can start and scale in response to increased load. This scenario seeks to determine the speed and efficiency of each application in becoming fully operational within a cloud environment.

These scenarios, structured by the framework, provide a comprehensive basis for evaluating the performance of each application in practical, high-demand conditions.

4.2.2.2 Execution of the PoC

The PoC was executed by deploying both applications within a Kubernetes cluster, utilizing Prometheus and Grafana for monitoring. Each scenario was run over a period of 20 minutes to collect a sufficient amount of data for robust analysis. This method ensured that the performance

of both Java (Quarkus) and Golang (Fiber) was observed under consistent and controlled conditions, allowing for a fair comparison between the two technologies.

4.2.2.3 Monitoring Metrics

Throughout the PoC, key performance metrics were collected using Prometheus and Grafana dashboards. The metrics included CPU usage, memory consumption, latency, and request handling capacity. This data was meticulously logged and analyzed to ensure the accuracy and traceability of the results.

The monitoring process was critical to understanding the nuanced performance differences between Java (Quarkus) and Golang (Fiber). It also provided the necessary insights to determine which technology was better performance suited for deployment in a Kubernetes-based cloud environment.

4.2.3 Make Decision

This section synthesizes the insights gathered during the study case implementation, focusing on evaluating the performance metrics of Java (Quarkus) and Golang (Fiber) applications within a Kubernetes environment. The goal is to provide a data-driven recommendation based on the observed results, ensuring that the decision is aligned with the objectives of performance, scalability, and cost-efficiency in cloud-native applications.

4.2.4 Analysis of Metrics

The analysis revealed key differences in the performance characteristics of Java (Quarkus) and Golang (Fiber):

- **Latency:** Golang consistently showed lower latency, particularly in the p99 percentile, indicating better performance under high load conditions.
- **CPU Usage:** Java had higher CPU usage, especially during the startup phase, but managed to equalize under load.

- **Memory Usage:** Both applications had comparable memory usage under load, though Java consumed more during the startup phase.
- **Startup Time:** Golang had a faster startup time, which is crucial for scaling in cloud environments.
- **Image Size:** Golang's Docker image was significantly smaller, allowing for faster deployment and scaling in cloud environments.

4.2.5 Final Decision

Based on the analysis, Golang (Fiber) demonstrated superior performance in terms of latency, CPU efficiency, and startup time, making it the preferred choice for cloud-native applications that require fast scaling and high efficiency. Java (Quarkus), while performant, had higher resource usage and slower startup times, which may impact cost efficiency in a cloud environment.

The final recommendation is to adopt Golang (Fiber) for scenarios where performance and efficiency are critical, especially in dynamic cloud environments. Java (Quarkus) remains a viable option for organizations already invested in the Java ecosystem, particularly if they value the extensive libraries and tooling available within the Java landscape.

4.3 Conclusion

The application of the framework in this case study identified and addressed gaps in the original PoC, enhancing key planning stages that were not explicitly detailed in the initial study Putra (2024). For instance, the framework highlighted areas such as stakeholder identification and detailed requirements specification, which were essential but initially overlooked.

In terms of **stakeholder analysis**, the framework facilitated the identification of teams and individuals impacted by the PoC. Although stakeholders were not explicitly named in the original study, analyzing the data in Table III-1 allowed the framework to infer key participants.

For **requirement specification**, the framework refined the PoC's requirements. While the original study focused mainly on non-functional aspects like scalability and efficiency, applying the framework revealed additional functional requirements such as CPU monitoring, latency monitoring, and startup time evaluation. This deeper analysis surfaced elements not previously identified.

The framework also emphasized the importance of **prerequisite identification**, leading to a thorough examination of necessary tools like Prometheus, Grafana, AWS, PostgreSQL, Kubernetes, and a GitHub repository. These tools, while essential, were not explicitly mentioned in the original PoC. By focusing on requirement analysis, the framework underscored the need for a well-defined setup phase.

In identifying **success measures and metrics**, the framework uncovered three additional success measures: latency performance, resource efficiency, and scalability. New metrics were also revealed, including image size, file operation latency, and database operation latency—each closely linked to functional requirements and success measures, providing a more robust evaluation of the PoC's outcomes.

The framework contributed to **timeline structuring** by organizing the PoC process into four stages: environment preparation, baseline metric analysis, stress testing under high load, and final metric analysis. This structure ensured a methodical and logical progression, with each stage building upon the last.

Additionally, **scenario development** benefited from the framework by expanding the testing scenarios beyond those in the original study. It led to the creation of a new scenario focused on the impact of image size on startup time and scalability, directly addressing resource efficiency requirements and success measures, thus broadening the comprehensiveness and relevance of the PoC.

In summary, the framework greatly enhanced the reverse engineering of the PoC by providing a systematic approach to identifying gaps, refining requirements, and improving the overall quality

of analysis. This methodology ensured that all critical aspects of the PoC were fully addressed, resulting in a more complete and effective evaluation.

Reflecting on the broader implications, this study suggests that the framework can serve as a valuable tool for improving PoC evaluations across various software engineering contexts. Its replicable methodology offers adaptability to different technologies and scenarios, enhancing the consistency and reliability of PoC outcomes.

CHAPTER 5

DISCUSSION

This chapter presents an in-depth discussion of the findings from the research, interpreting the results in the context of existing literature and identifying key contributions to the field of software engineering, particularly in Proof of Concept (PoC). The aim of this discussion is demonstrating how the proposed framework fills gaps identified in previous studies and exploring the broader implications for the field.

5.1 Research Questions

This section addresses the research questions formulated at the outset of this work, providing detailed responses based on the findings from the study.

5.1.1 RQ1: What is the definition of Proof of Concept (PoC) in the context of Software Engineering

The first research question sought to establish a clear definition of PoCs within software engineering. This research provides a refined definition that addresses the conceptual ambiguity found in the literature. A PoC in software engineering is defined as a strategic process designed to validate the feasibility and effectiveness of a proposed solution. It involves controlled experiments to assess technical viability, explore unknowns, and inform decision-making during the early stages of software development.

This study defines Proof of Concept (PoC) as a targeted, time-constrained exercise aimed at reducing technical risks, supporting informed decision-making, and validating specific concepts before committing substantial resources to full-scale development. It is important to distinguish PoCs from prototypes or pilot projects, which typically have broader objectives and are often more expansive in scope.

5.1.2 RQ2: What are the key factors, processes, and practices that contribute to the planning, execution, and evaluation of PoCs in software engineering

In response to RQ2, important key factors, processes, and practices emerged in Proofs of Concept (PoC) experiments. Key factors include a clear understanding of the PoC's objectives and stakeholder alignment, assessing the experiment's feasibility. The PoC code is generally not intended for use in the final product and should be discarded after completion due to its limited scope and short duration. These characteristics reinforce the temporary nature of the PoC, promoting risk mitigation, support for learning, feasibility assessment, and assistance in the decision-making process.

The execution of PoCs is structured around three primary processes: planning, execution, and decision-making. These processes interact continuously, allowing the PoC to adapt dynamically based on emerging insights. During execution, new requirements or metrics often arise, feeding back into the planning and execution phases. This iterative feedback mechanism enables the PoC to remain aligned with its objectives while continuously refining its approach and scope as needed.

Regarding practices, this study recommends implementing the three main PoC processes - planning, execution, and decision-making - in small, incremental steps. This approach provides flexibility, allowing for real-time adjustments and fostering continuous learning throughout the PoC. Each step helps ensure that the PoC effectively meets its objectives and supports informed decision-making within software engineering.

5.1.3 RQ3: What are the benefits and challenges of a framework for PoC execution in software engineering

The development of a structured framework for PoC execution offers benefits while also presenting challenges, both of which were addressed throughout this research. The framework enhances consistency and clarity in the PoC process by providing a detailed, step-by-step guide for practitioners. It aids teams in managing resources effectively and ensures that PoCs align

with project goals. Additionally, by establishing clear evaluation criteria, the framework supports decision-making and increases the project success.

However, some challenges emerge, such as the difficulty of generalizing the framework to all software project types, given the diversity in development methodologies and project scales. Another challenge is the need for careful management to avoid scope creep during the PoC phase, which can compromise the PoC's focus and lead to inefficiency.

To address these challenges, the proposed framework incorporates feedback loops and clearly defined objectives, though further research is required to adapt the framework to a broader range of contexts.

5.2 Key Findings Interpretation

The results presented in Chapters 4 and 5 reveal contributions of the PoC framework. The framework enables an **structured planning for PoCs** in software engineering projects, addressing the need for clearer guidelines in stakeholder analysis, success metrics, and scenario development.

The framework **offers a systematic approach** to identifying and involving key stakeholders, which had previously been a point of weakness in the two case studies analyzed. This approach led to better team alignment with project goals. Additionally, by **aligning metrics** with **success measures** - and further aligning these measures with **requirements** gathered from stakeholders - the framework has shown an improvement in the **robustness of PoC analysis**. This alignment helps that the evaluation of PoC outcomes is more meaningful and targeted.

The framework also emphasizes the importance of **gathering prerequisites** as a important step. This step is necessary to carry out the PoC, helping to avoid potential delays during execution. Finally, by **enhancing scenario** development the framework minimizes potential issues. Collectively, these improvements demonstrate how the framework enhances the PoCs execution, reducing ambiguity throughout the planning and execution phases.

Moreover, this research advances the current body of literature by providing a structured and detailed framework for conducting PoCs in software engineering. The proposed framework offer specific, actionable steps for each phase of the PoC process, bridging the gap between academic theory and real-world application. Moreover, the framework incorporates elements from both academic and grey literature, providing a more holistic approach to PoC implementation.

5.3 Limitations of the Study

This research provides valuable insights into PoC processes in software engineering, though certain limitations merit attention. First, as in any qualitative study, **potential biases** in data collection and interpretation may have influenced the findings. Despite efforts to minimize bias through systematic analysis, the selection of case studies and literature could still impact the conclusions drawn. Additionally, while the literature review aimed for thoroughness, **it may not have been exhaustive** emerging studies or unpublished work might not have been captured, potentially narrowing the scope of the findings. As the field progresses, continuous review and refinement of the framework will be essential to integrate new advancements.

Furthermore, PoC implementation may involve significant time, expertise, and budget allocations, which may not be feasible in for the PoC project. **Teams with limited resources** could face challenges in fully utilizing the framework's processes, as its applicability may vary significantly based on organizational capacity and available project budgets.

These limitations highlight areas where further research is necessary to strengthen the framework's robustness and extend its applicability across diverse contexts within software engineering.

5.4 Implications for Practice

The findings from this research offer significant practical implications for software engineering professionals. The framework provides a structured approach to planning and executing PoCs, ensuring that key factors such as stakeholder engagement, success metrics, and scenario planning are adequately addressed. This contributes to resource allocation, improved risk management,

and ultimately, higher project success rates. Additionally, by clearly defining each stage of the PoC process, the framework helps practitioners avoid common pitfalls such as scope creep, misaligned objectives, and inadequate testing.

5.5 Recommendations for Future Research

Future research should focus on further validating the framework in live software development projects. This would provide real-time feedback on its effectiveness and allow for refinements based on practical experiences. A key area of study could involve comparing the success rates of PoC projects executed with and without the framework. By conducting controlled experiments, live case studies, and structured interviews with project teams, researchers could quantify the impact of the framework on PoC success. Metrics such as the percentage of projects that meet their technical and business objectives, as well as resource efficiency and stakeholder satisfaction, could be used to assess its effectiveness. Surveys with software engineering professionals could further provide valuable insights into the practical challenges and benefits experienced during PoC execution.

Additionally, the framework could be expanded to include emerging technologies, which present unique challenges and opportunities for PoC implementation. Technologies such as artificial intelligence, blockchain, and cloud-native systems may require adaptations to the framework. Conducting longitudinal studies across different industries would also help assess the framework's applicability and identify any necessary adaptations for various project types. These studies, which could involve surveys, interviews, and case studies, would contribute to a more comprehensive understanding of the framework's effectiveness across a wide range of contexts and help refine its scope.

5.6 Conclusion

This discussion chapter has highlighted the key contributions of the proposed PoC framework to the field of software engineering, including its practical applicability, potential limitations,

and areas for future research. By providing a structured approach to PoC implementation, this research offers both theoretical insights and practical tools for improving the success of software projects. While further validation and expansion of the framework are necessary, the findings represent a significant step forward in the understanding and execution of PoCs in software development.

CONCLUSION AND RECOMMENDATIONS

This thesis addresses the ambiguity surrounding the definition and implementation of Proof of Concept (PoC) in software engineering by offering both a clear, structured definition and a comprehensive framework. This framework aids practitioners in the planning, execution, and evaluation of PoCs, providing a practical tool for PoC practitioners. By examining existing literature and real-world practices, this research clarifies PoC characteristics and processes, emphasizing the importance of setting clear objectives, managing risks, and ensuring technical validation. Additionally, it distinguishes PoCs from other early-stage artifacts, such as MVPs and prototypes, reducing confusion between these concepts and enhancing clarity for practical application.

This thesis address the ambiguity surrounding the definition and implementation of Proof of Concept (PoC) in software engineering, offering a structured framework that aids practitioners in planning, executing, and evaluating PoCs. By examining existing literature and real-world practices, this research clarified PoC characteristics and processes, emphasizing the importance of setting clear objectives, managing risks, and ensuring technical validation. Additionally, it distinguishes PoCs from other early-stage artifacts, such as MVPs and prototypes, avoiding confusion between these concepts.

To achieve this, the research focused on answering three central questions: (1) What defines a PoC in the context of software engineering? (2) What key characteristics and processes ensure effective PoC planning, execution, and evaluation? (3) What are the benefits and challenges of implementing a structured PoC framework? By addressing these questions, this study provided a comprehensive understanding of the PoC process, offering professionals insights that enhance decision-making and project outcomes.

The case study in the banking domain shows the framework's adaptability and effectiveness, especially in environments where regulatory compliance, risk mitigation, and precise requirements

management are critical. The structured approach improved stakeholder identification, requirement specification, metric selection, timeline organization, and the creation of test scenarios. These improvements reflect the framework's potential to increase the consistency, reliability, and relevance of PoC outcomes across various industries and technical landscapes.

The application of this framework to the comparative PoC of Java and Golang in Kubernetes environments demonstrated its value in conducting thorough performance evaluations. By defining metrics such as startup time, latency, resource efficiency, and scalability, the framework enabled a detailed comparison that clarified the nuanced performance and operational suitability of each technology. Additionally, the framework helped uncover new requirements, including CPU monitoring, latency tracking, and startup time evaluation, which were crucial for an in-depth performance assessment. It also facilitated the identification of critical prerequisites, such as Prometheus, Grafana, AWS, PostgreSQL, Kubernetes, and a GitHub repository, ensuring all necessary tools were in place for seamless execution. The framework further defined three additional success measures—latency performance, resource efficiency, and scalability—that provided a more comprehensive basis for evaluating PoC outcomes. In summary, the framework significantly enhanced the reverse engineering of the PoC by systematically identifying gaps, refining requirements, and elevating the overall quality of the analysis. This structured approach ensured that all essential aspects of the PoC were addressed, resulting in a complete and effective evaluation.

Looking forward, future validation of this framework could benefit from experimental methods, including controlled experiments, structured interviews, surveys, and live case studies. Such methods would further assess the framework's utility and adaptability, applying it across diverse scenarios to refine its application and effectiveness. Additionally, future research could explore the framework's potential applicability to other fields beyond software engineering, further strengthening its flexibility and expanding its relevance.

In conclusion, this study has laid a foundational approach for a more systematic, transparent, and effective PoC process, providing software engineering practitioners with a practical tool for decision-making and project success.

APPENDIX I

GREY LITERATURE ASSESSMENT

1. Grey literature assessment quality for Performance benchmarking between Java and Golang in Kubernetes study case

Grey literature, while valuable for its practical insights, requires careful evaluation to ensure its credibility and relevance. To assess the quality of this case study, Garousi *et al.* (2019) provides a check list quality assessment for evaluating grey literature in software engineering. The assessment examines four fundamental aspects: **authority, purpose, methodology, and relevance**.

The **authority** assessment confirms the source's credibility through the presenter's professional background in software engineering. The presenter's experience with microservices and Kubernetes architecture provides the foundation for technical credibility in the source material.

The **purpose** assessment examines the content's educational objectives. The video presents a performance comparison between Java and Golang in Kubernetes environments, addressing a specific technical question that resonates with both research and industry applications.

The **methodological** assessment reveals both strengths and limitations in the presentation. While the video demonstrates benchmarking procedures, it lacks complete documentation of experimental parameters. The absence of detailed hardware specifications and test configurations limits the reproducibility of the demonstrated results.

The **relevance** assessment confirms the content's applicability to current software engineering practices. The video's focus on microservices and containerization aligns with current software architecture trends in both academic and industrial contexts.

The quality assessment implements a scoring system based on Garousi *et al.* (2019)'s framework: zero indicates unmet criteria, 0.5 represents partial fulfillment, and one signifies complete

satisfaction of requirements. This systematic approach provides an objective evaluation structure for industry-generated content.

Quality assessment checklist

- **Authority of the Producer:**
  - **Is the publishing organization reputable?** Yes. This is a YouTube channel that has been producing technical videos since 2020. The channel has over 60,000 subscribers and more than 4 million views, indicating a strong following and significant impact in the grey literature field. *Measure: 1*
  - **Is the author associated with a reputable organization?** Yes. The author is employed by Juniper Networks, a company founded in 1996 that has more than eleven thousand employees across fifty different countries. *Measure: 1*
  - **Has the author published other work in the field?** Yes. The author has created 227 technical videos since 2020. *Measure: 1*
  - **Does the author have expertise in the area?** Yes. The author has expertise as a Benchmark Engineer and has been working as a software engineer since 2011. *Measure: 1*
- **Methodology:**
  - **Does the source have a clearly stated aim?** Yes, the video aims to compare the performance of Java and Go applications in Kubernetes. *Measure: 1*
  - **Does the source have a stated methodology?** Yes, the methodology is described, including specific metrics (e.g., CPU and memory usage, latency) and the tools used (Prometheus, Grafana). *Measure: 1*
  - **Is the source supported by authoritative references?** No. The video does not cite other authoritative references. *Measure: 0*
  - **Are any limits clearly stated?** The limits of the benchmarks are implied (e.g., specific frameworks used, types of tests performed). *Measure: 1*
  - **Does the work cover a specific question?** Yes, it covers the specific question of performance comparison between two frameworks. *Measure: 1*

  - **Does the work refer to a particular population or case?** Yes, it refers to applications running in a Kubernetes environment, which is relevant to a specific technical context. *Measure: 1*

- **Objectivity:**
  - **Is the work balanced in presentation?** Yes, the case study provides both pros and cons of each technology. *Measure: 1*
  - **Is the statement in the sources as objective as possible?** Yes, the video seems to focus on presenting data rather than opinions. *Measure: 1*
  - **Is there vested interest?** No. Given the technical nature of the video and the metrics established, the video does not appear to have any vested interest. *Measure: 1*
  - **Are the conclusions supported by the data?** Yes, the conclusions are based on the performance metrics collected during the tests. *Measure: 1*
- **Date:**
  - **Does the item have a clearly stated date?** Yes, the video is dated August 17, 2024. *Measure: 1*
- **Position w.r.t. Related Sources:**
  - **Have key related GL or formal sources been linked to/discussed?** No. There is no mention of related literature or sources in the video. *Measure: 0*
- **Novelty:**
  - **Does it enrich or add something unique to the research?** Yes. The case study provides new data points and comparisons that are relevant to practitioners. *Measure: 1*
  - **Does it strengthen or refute a current position?** Yes. It provides practical insights that can strengthen or refute positions on the two technologies analyzed. *Measure: 1*
- **Impact:**
  - **Impact metrics:** Yes. The case study has a significant number of views, comments, and shares, indicating a high impact in the grey literature space. *Measure: 1*
- **Outlet Type:**
  - **2nd tier GL:** Potentially, as it is a video providing a technical analysis. *Measure: 0.5*

Based on the quality assessment criteria Garousi *et al.* (2019), this source would be considered trustworthy. The source has a total of 17.5 points out of 20. According to Garousi *et al.* (2019) criteria, this source can be considered particularly reliable for practical insights and technical comparisons in the field of software engineering.

APPENDIX II

POC FRAMEWORK SCREENSHOTS: FRAMEWORK EVALUATION: A CASE STUDY OF DATABASE VERSIONING IN THE BANKING DOMAIN

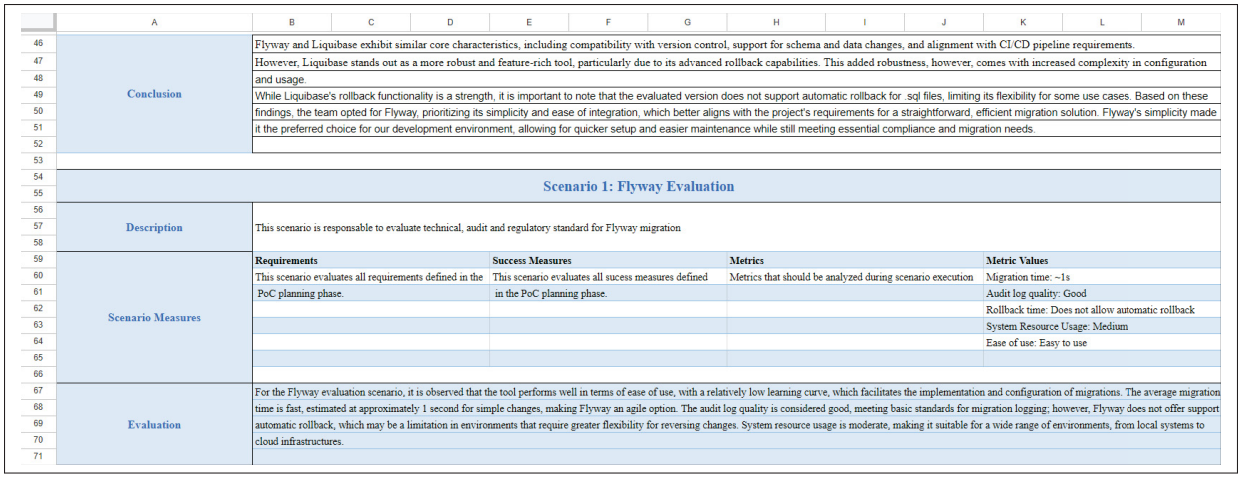
0.1 Annexed figures

POC Product / Project Name												Date
Evaluate database versioning tool												08/07/2024
Determine Purpose / Objective	This section is dedicated to outlining the PoC objectives, which must be aligned with the assumptions that motivated its creation.						Team	Role		Person Identification		
								Database Team				
								Business Team				
								Software developers				
								Managers				
Characteristics	Assessment and Decision Making											
	Determine Feasibility											
	Solve learning needs											
	Evaluate risks											
	Discover unknowns											
	Identify Gaps											
	Analyze techinical limitations											
	Determine Requirements						Determine Prerequisites					
	The tool must have version control for schema and data changes.						Access to Confluence					
	The tool must have rollback mechanism and compatibility with Oracle and PostgreSql databases.						Isolated database instances running in the local environment, aligned with test requirements.					
	The tool should minimize system downtime to ensure continuous availability of services.						Administrator privileges on the workstation.					
	The tool must be optimized to maintain performance under high transaction loads						Network and database access in test environnement for executing and validating test scenarios.					
	The toll should be compatible with the existing CI/CD pipeline						Valid Docker license to support setup and testing processes.					

Figure-A II-1 Framework session: Objective, Team, Characteristics, Requeriments and Prerequisites

	Determine Success Measures						Determine Metrics					
	Migration application time, measured against documented performance benchmarks from past migrations.						Migration time (seconds/minutes): compared against past performance benchmarks.					
	Completeness and clarity of audit logs, evaluated for compliance with internal documentation standards.						Audit log quality: evaluated according to compliance standards from Confluence documentation.					
	Effectiveness of rollback procedures for consistency and reliability.						Rollback time: measured by the time required to revert to a previous state.					
							System Resource Usage: CPU and memory consumption					
							Ease of use: Ease of integration with Java applications.					
Timeline	Stage 1				Stage 2		Stage 3				Stage 4	
	Environment setup and preparation for migration scripts				Execution of scenarios using Flyway		Executuin of scearios using Liquibase				Data analysis and recommendation	
	Environment setup and preparation of migration scripts				Execute test scenarios and record measure metrics.		Execute test scenarios and record measure metrics.				Analysis of the collected metrics and developers avaliation to assess which solution integrates best with the existing Java development environment	

Figure-A II-2 Framework session: Success measures, metrics and timeline

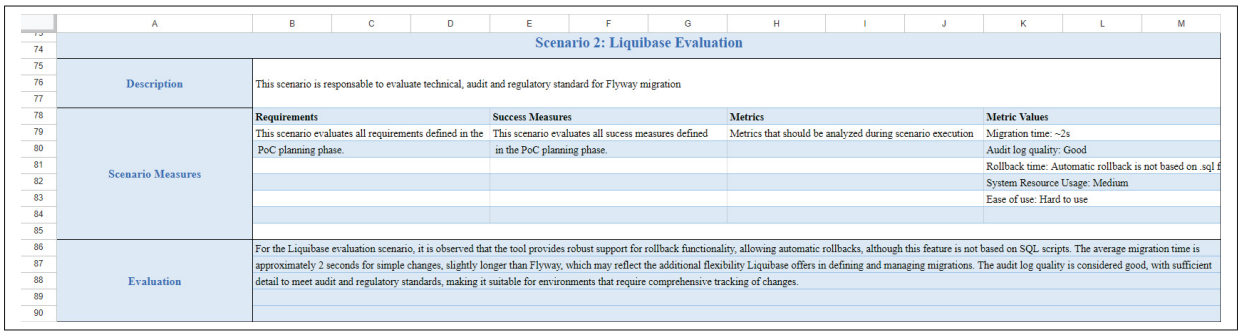


Conclusion	Flyway and Liquibase exhibit similar core characteristics, including compatibility with version control, support for schema and data changes, and alignment with CI/CD pipeline requirements. However, Liquibase stands out as a more robust and feature-rich tool, particularly due to its advanced rollback capabilities. This added robustness, however, comes with increased complexity in configuration and usage. While Liquibase's rollback functionality is a strength, it is important to note that the evaluated version does not support automatic rollback for .sql files, limiting its flexibility for some use cases. Based on these findings, the team opted for Flyway, prioritizing its simplicity and ease of integration, which better aligns with the project's requirements for a straightforward, efficient migration solution. Flyway's simplicity made it the preferred choice for our development environment, allowing for quicker setup and easier maintenance while still meeting essential compliance and migration needs.

Scenario 1: Flyway Evaluation

Description	This scenario is responsable to evaluate technical, audit and regulatory standard for Flyway migration			
Scenario Measures	**Requirements**	**Success Measures**	**Metrics**	**Metric Values**
	This scenario evaluates all requirements defined in the PoC planning phase.	This scenario evaluates all sucess measures defined in the PoC planning phase.	Metrics that should be analyzed during scenario execution	Migration time: ~1s
				Audit log quality: Good
				Rollback time: Does not allow automatic rollback
				System Resource Usage: Medium
				Ease of use: Easy to use
Evaluation	For the Flyway evaluation scenario, it is observed that the tool performs well in terms of ease of use, with a relatively low learning curve, which facilitates the implementation and configuration of migrations. The average migration time is fast, estimated at approximately 1 second for simple changes, making Flyway an agile option. The audit log quality is considered good, meeting basic standards for migration logging; however, Flyway does not offer support automatic rollback, which may be a limitation in environments that require greater flexibility for reversing changes. System resource usage is moderate, making it suitable for a wide range of environments, from local systems to cloud infrastructures.			

Figure-A II-3 Framework session: Poc conclusion and scenario 1 description



Scenario 2: Liquibase Evaluation

Description	This scenario is responsable to evaluate technical, audit and regulatory standard for Flyway migration			
Scenario Measures	**Requirements**	**Success Measures**	**Metrics**	**Metric Values**
	This scenario evaluates all requirements defined in the PoC planning phase.	This scenario evaluates all sucess measures defined in the PoC planning phase.	Metrics that should be analyzed during scenario execution	Migration time: ~2s
				Audit log quality: Good
				Rollback time: Automatic rollback is not based on .sql f
				System Resource Usage: Medium
				Ease of use: Hard to use
Evaluation	For the Liquibase evaluation scenario, it is observed that the tool provides robust support for rollback functionality, allowing automatic rollbacks, although this feature is not based on SQL scripts. The average migration time is approximately 2 seconds for simple changes, slightly longer than Flyway, which may reflect the additional flexibility Liquibase offers in defining and managing migrations. The audit log quality is considered good, with sufficient detail to meet audit and regulatory standards, making it suitable for environments that require comprehensive tracking of changes.			

Figure-A II-4 Framework session: Scenario 2 description

APPENDIX III

POC FRAMEWORK SCREENSHOTS: PERFORMANCE BENCHMARKING CASE STUDY OF JAVA AND GOLANG IN KUBERNETESS

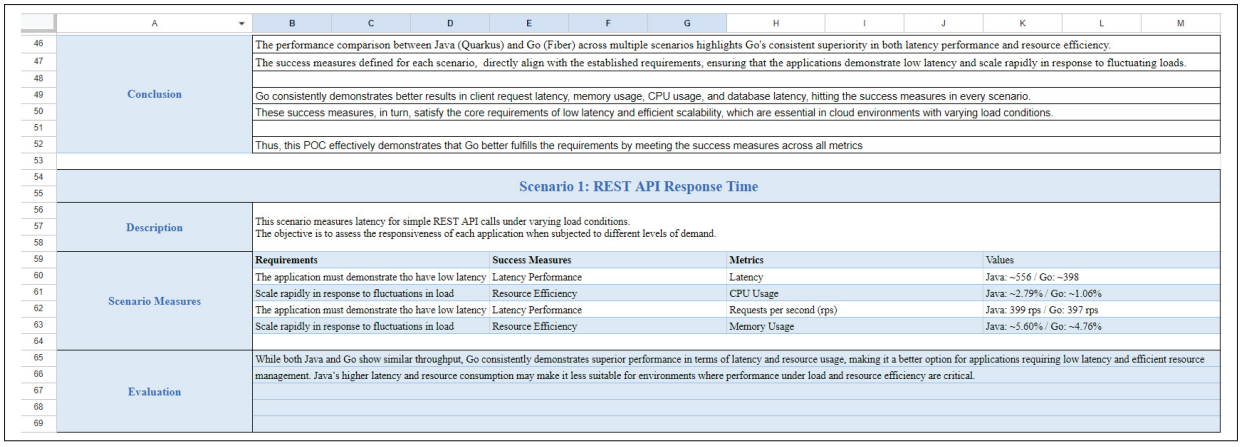
0.1 Annexed figures

POC Product / Project Name												Date
Evaluate performance: Java (Quarkus) vs Golang (Fiber)												30/09/2024
Determine Purpose / Objective	Evaluate the performance characteristics of Java (Quarkus) and Golang (Fiber) applications within Kubernetes						**Team**	Role			Person Identification	
								DevOps teams				
								Software architects				
								Software developers				
Characteristics	**Assessment and Decision Making**						**Development and Experimentation**					
	Risk Mitigation											
	Decision Support											
	Helps to choose / evaluate technology											
	Test requirements											
	Determine Requirements						**Determine Prerequisites**					
	The application must demonstrate tho have low latency						Access to Prometheus and Grafana for monitoring and logging.					
	The application must demonstrate the ability to scale rapidly in response to fluctuations in load						Access to an S3 bucket in AWS for file storage.					
	Resource usage should be optimized to minimize costs within the cloud environment.						Read and write access to a PostgreSQL database server.					
							Access to a Kubernetes cluster with appropriate configurations.					
							A GitHub repository containing the source code for both Java and Golang applications.					

Figure-A III-1 Framework session: Objective, Team, Characteristics, Requeriments and Prerequisites

	Determine Success Measures						**Determine Metrics**					
	Latency Performance: Measured with an emphasis on the system's responsiveness under increasing load condition						Startup Time: The time taken for the application to become ready in a Kubernetes pod, measured from pod scheduling to readiness.					
	Resource Efficiency: Monitoring and minimizing CPU and memory usage during both idle and active states, ensuring low resource consumption during non-peak periods while maintaining or improving overall performance.						CPU Usage: CPU usage during idle and load states, with a focus on both the average and peak usage.					
	Scalability: Evaluated by the application's ability to scale efficiently in response to load changes, including the time required to start and become ready after a scale event, which is influenced by factors such as application image size.						Memory Usage: Memory consumption, with a focus on differences between idle and active states, and how these variations impact performance.					
							Latency: Measure latency between request / response					
							Request Handling Capacity: The number of requests per second (RPS) handled by each application under varying load conditions					
							Image Size: The size of the Docker images used for each technology					
							File Operation Latency: The time taken for file operations, including reading from the local file system and uploading to S3					
							atabase Operation Latency: The time required to write to the database					
Timeline	**Stage 1**						**Stage 2**				**Stage 3**				**Stage 4**
	Setup of Prerequisites						**Baseline Performance Testing**				**Load Testing**				**Analysis and Documentation**
	Setting up all necessary prerequisites, such as						Assess the behavior of the Java (Quarkus) and				Executing load tests and collect metrics				Analyzing the collected data, comparing the
	configuring the S3 bucket for storage						Golang (Fiber) applications in an idle state,								performance of the Java (Quarkus) and Golang (Fiber)
	ensuring access to Prometheus and Grafana and						focusing on initial CPU and memory usage,								applications across all measured metrics,
	preparing the Kubernetes cluster						startup times, and other key metrics								documenting the findings and make decision

Figure-A III-2 Framework session: Success measures, metrics and timeline



Conclusion	The performance comparison between Java (Quarkus) and Go (Fiber) across multiple scenarios highlights Go's consistent superiority in both latency performance and resource efficiency. The success measures defined for each scenario, directly align with the established requirements, ensuring that the applications demonstrate low latency and scale rapidly in response to fluctuating loads. Go consistently demonstrates better results in client request latency, memory usage, CPU usage, and database latency, hitting the success measures in every scenario. These success measures, in turn, satisfy the core requirements of low latency and efficient scalability, which are essential in cloud environments with varying load conditions. Thus, this POC effectively demonstrates that Go better fulfills the requirements by meeting the success measures across all metrics

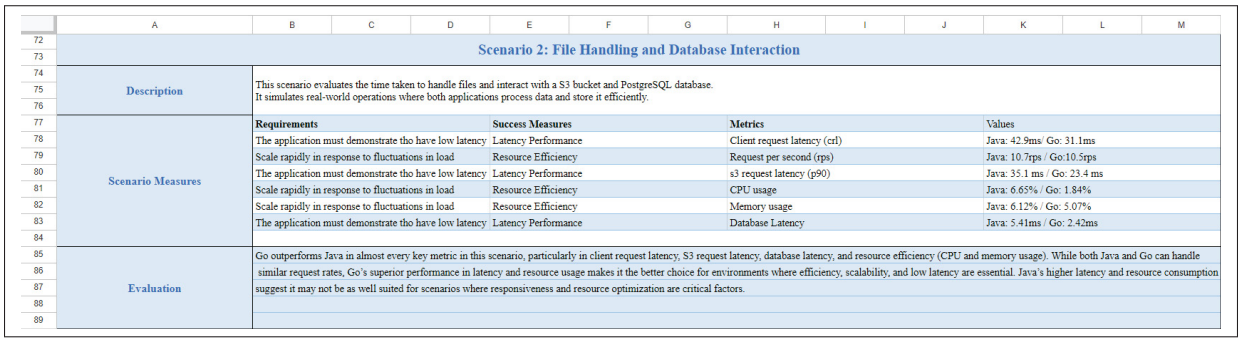
Scenario 1: REST API Response Time

Description	This scenario measures latency for simple REST API calls under varying load conditions. The objective is to assess the responsiveness of each application when subjected to different levels of demand.

Scenario Measures				
Requirements	**Success Measures**	**Metrics**	Values
The application must demonstrate tho have low latency	Latency Performance	Latency	Java: ~556 / Go: ~398
Scale rapidly in response to fluctuations in load	Resource Efficiency	CPU Usage	Java: ~2.79% / Go: ~1.06%
The application must demonstrate tho have low latency	Latency Performance	Requests per second (rps)	Java: 399 rps / Go: 397 rps
Scale rapidly in response to fluctuations in load	Resource Efficiency	Memory Usage	Java: ~5.60% / Go: ~4.76%

Evaluation	While both Java and Go show similar throughput, Go consistently demonstrates superior performance in terms of latency and resource usage, making it a better option for applications requiring low latency and efficient resource management. Java's higher latency and resource consumption may make it less suitable for environments where performance under load and resource efficiency are critical.

Figure-A III-3 Framework session: Poc conclusion and scenario 1 description



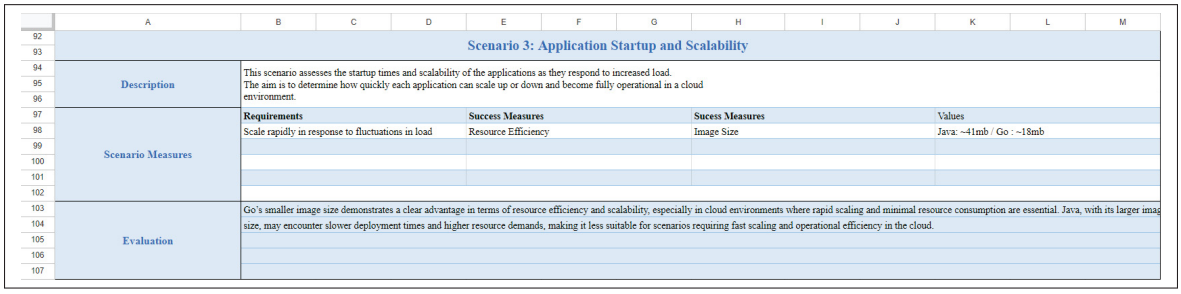
Scenario 2: File Handling and Database Interaction

Description	This scenario evaluates the time taken to handle files and interact with a S3 bucket and PostgreSQL database. It simulates real-world operations where both applications process data and store it efficiently.

Scenario Measures			
Requirements	**Success Measures**	**Metrics**	Values
The application must demonstrate tho have low latency	Latency Performance	Client request latency (crl)	Java: 42.9ms/ Go: 31.1ms
Scale rapidly in response to fluctuations in load	Resource Efficiency	Request per second (rps)	Java: 10.7rps / Go:10.5rps
The application must demonstrate tho have low latency	Latency Performance	s3 request latency (p90)	Java: 35.1 ms / Go: 23.4 ms
Scale rapidly in response to fluctuations in load	Resource Efficiency	CPU usage	Java: 6.65% / Go: 1.84%
Scale rapidly in response to fluctuations in load	Resource Efficiency	Memory usage	Java: 6.12% / Go: 5.07%
The application must demonstrate tho have low latency	Latency Performance	Database Latency	Java: 5.41ms / Go: 2.42ms

Evaluation	Go outperforms Java in almost every key metric in this scenario, particularly in client request latency, S3 request latency, database latency, and resource efficiency (CPU and memory usage). While both Java and Go can handle similar request rates, Go's superior performance in latency and resource usage makes it the better choice for environments where efficiency, scalability, and low latency are essential. Java's higher latency and resource consumption suggest it may not be as well suited for scenarios where responsiveness and resource optimization are critical factors.

Figure-A III-4 Framework session: Scenario 2 description



Scenario 3: Application Startup and Scalability

Description	This scenario assesses the startup times and scalability of the applications as they respond to increased load. The aim is to determine how quickly each application can scale up or down and become fully operational in a cloud environment.

Scenario Measures			
Requirements	**Success Measures**	**Sucess Measures**	Values
Scale rapidly in response to fluctuations in load	Resource Efficiency	Image Size	Java: ~41mb / Go : ~18mb

Evaluation	Go's smaller image size demonstrates a clear advantage in terms of resource efficiency and scalability, especially in cloud environments where rapid scaling and minimal resource consumption are essential. Java, with its larger image size, may encounter slower deployment times and higher resource demands, making it less suitable for scenarios requiring fast scaling and operational efficiency in the cloud.

Figure-A III-5 Framework session: Scenario 3 description

0.2 Tables in annexes

Table-A III-1 PoC Team Members based on Analyzed References

PoC	Team Members Involved in the PoC
Wasonga (2023)	Key users, analysts, technical teams
Lindsey (2023)	Vulnerability Management Teams, IT personnel (Configuration Management, Patching, Asset Inventory), GRC Teams, (Optional) GRC-aligned Legal, Brand Protection, Privacy Teams, Internal Offensive Penetration Testing and Red Teams, Security Operations Teams, Incident Response Teams, Cyber Threat Intelligence, Hunting, and Research Teams
Yeung & Bull (2022) /	Business owners, AI developers, data scientists, IT, SecOps teams, AI software providers
National Archives and Records Administration (2019)	Diverse range of job roles, from administrative support to senior management, including network and desktop engineers, network/software support personnel, security engineers, trainers, administrators
Atlassian (2023)	Product teams, clients, stakeholders
Forbes Technology Council (2024)	Functional, technical, marketing teams, customers
Genway, Neelakantaiah, Cruise & Roberts (2022)	Not clearly described
Capgemini (2023)	Not clearly described
Rouse (2024)	Employees or team leaders, project managers, stakeholders, the product owner, managers, or investors
Bigelow (2024)	Users, stakeholders, investors, project team
TechTarget Contributor (2024)	Project managers, development teams, technical teams, other relevant stakeholders
Aryan (2023)	Not clearly described
Ashraf (2023)	Users, business customers, technical team
Orient Software (2023)	Project Manager, Product Owner, Team Leader, Tech Leader, Software Developer, UI/UX Designer, Quality Assurance (QA)
Netguru (2021)	Stakeholders, product team, users, technical team
Recapped.io (2022)	Customers, leadership, marketing, sales, product team, consultants, decision-makers, stakeholders
GeeksforGeeks (2024)	Users, stakeholders, development team, QA engineers, software architects
Invensis Learning (2024)	Decision-makers, technical experts, project stakeholders, other relevant parties
TechnologyAdvice (2023)	Project management team, other relevant professionals

BIBLIOGRAPHY

Aeronautics, U. S. N., of Scientific, S. A. O., Information, T., Aeronautics, U. S. N., Scientific, S. A., Division, T. I., Aeronautics, U. S. N., Scientific, S. A., Office, T. I., Aeronautics, U. S. N., Scientific, S. A., Branch, T. I., Scientific, N., Facility, T. I., Aeronautics, U. S. N., Scientific, S. A., Program, T. I. & for AeroSpace Information, N. C. (1995). *Scientific and Technical Aerospace Reports*. NASA, Office of Scientific and Technical Information. Retrieved from: https://books.google.ca/books?id=TShgtOP4FNgC.

Anderson, G. (2023). *Design Thinking for Tech: Solving Problems and Realizing Value in 24 Hours*. Pearson Education. Retrieved from: https://books.google.ca/books?id=8bUtzwEACAAJ.

Aryan, Y. (2023). Creating a Proof of Concept: A Practical Guide for Software Development [Web page]. Retrieved from: https://www.linkedin.com/pulse/creating-proof-concept-practical-guide-software-yahya-aryan/.

Ashraf, S. (2023). 7-Step Best Practices for a Successful Proof of Concept (POC) [Web page]. Retrieved from: https://www.linkedin.com/pulse/7-step-best-practices-successful-proof-concept-poc-shehzad/.

Atlassian. (2023). Proof of Concept in Product Development [Web page]. Retrieved from: https://www.atlassian.com/work-management/project-management/proof-of-concept.

Bales, K. (1981). *Structures and Dynamics Division Research and Technology Plans, Fiscal Year, 1981*. Retrieved from: https://books.google.ca/books?id=AXBFAQAAIAAJ.

Baptista, G. & Abbruzzese, F. (2024). *Software Architecture with C# 12 and .NET 8: Build Enterprise Applications Using Microservices, DevOps, EF Core, and Design Patterns for Azure*. Packt Publishing Limited. Retrieved from: https://books.google.ca/books?id=wJJQ0AEACAAJ.

Bell, M. (2023). *Software Architect*. Wiley. Retrieved from: https://books.google.ca/books?id=-ryvEAAAQBAJ.

Beningo, J. (2022). *Embedded Software Design: A Practical Approach to Architecture, Processes, and Coding Techniques*. Apress. Retrieved from: https://books.google.ca/books?id=cIoPzwEACAAJ.

Bigelow, S. J. (2024). How to kickstart a proof of concept IT project [Web page]. Retrieved from: https://www.techtarget.com/searchitoperations/tip/How-to-kickstart-a-proof-of-concept-IT-project.

Brisals, S. & Hedger, L. (2024). *Serverless Development on AWS*. O'Reilly Media. Retrieved from: https://books.google.ca/books?id=afDvEAAAQBAJ.

Burns, B. (2018). *Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services*. O'Reilly Media. Retrieved from: https://books.google.ca/books?id=5hJNDwAAQBAJ.

Caelen, O. & Blete, M. (2023). *Developing Apps with GPT-4 and ChatGPT*. O'Reilly Media. Retrieved from: https://books.google.ca/books?id=XaXTEAAAQBAJ.

Capgemini. (2023). Why Are There So Many Proof-of-Concepts (POCs)? [Web page]. Retrieved from: https://www.capgemini.com/no-no/insights/expert-perspectives/why-are-there-so-many-proof-of-concepts-pocs/.

Codica. (2024). PoC vs MVP: What's the Difference and How to Choose? [Blog post]. Retrieved from: https://www.codica.com/blog/poc-vs-mvp/.

Collins. [Collins English Dictionary, 5th edition]. (2024). Definition of proof of concept. HarperCollins Publishers. Retrieved on 2024-03-14 from: https://www.collinsdictionary.com/dictionary/english/proof-of-concept.

Deckert, J. C. & Szalai, K. J. (1954). ANALYTIC REDUNDANCY MANAGEMENT FOR FLIGHT CONTROL SENSORS. *AD A0132 910 ADVANCES IN SENSORS AND THEIR INTEGRATION INTO AIRCRAFT 1*, 9, 2.

Enríquez, R. & Salazar, A. (2018). *Software Architecture with Spring 5.0: Design and architect highly scalable, robust, and high-performance Java applications*. Packt Publishing. Retrieved from: https://books.google.ca/books?id=VvFsDwAAQBAJ.

Fairbanks, G. (2010). *Just Enough Software Architecture: A Risk-Driven Approach*. Marshall & Brainerd. Retrieved from: https://books.google.ca/books?id=5UZ-AQAAQBAJ.

Fernando, C. (2022). *Solution Architecture Patterns for Enterprise: A Guide to Building Enterprise Software Systems*. Apress. Retrieved from: https://books.google.ca/books?id=I0RgzwEACAAJ.

Forbes Technology Council. (2024). How To Craft A Solid Proof Of Concept That Gets Your Message Across [Web page]. Retrieved from: https://www.forbes.com/sites/forbestechcouncil/2024/05/14/how-to-craft-a-solid-proof-of-concept-that-gets-your-message-across/.

Ford, N., Parsons, R. & Kua, P. (2017). *Building Evolutionary Architectures: Support Constant Change*. O'Reilly Media. Retrieved from: https://books.google.ca/books?id=pYI2DwAAQBAJ.

Fowler, M. (2015). Sacrificial Architecture [Blog post]. Retrieved from: https://martinfowler.com/bliki/SacrificialArchitecture.html.

Fulton, R. (1985). *IPAD: A Unique Approach to Government/industry Cooperation for Technology Development and Transfer*. School of Engineering and Applied Science, George Washington University. Retrieved from: https://books.google.ca/books?id=OnVGAQAAIAAJ.

Furht, B. (1998). *Handbook of Multimedia Computing*. Taylor & Francis. Retrieved from: https://books.google.ca/books?id=Mn0jNqzVsDgC.

Garousi, V., Felderer, M. & Mäntylä, M. V. (2019). Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology*, 106, 101-121. doi: https://doi.org/10.1016/j.infsof.2018.09.006.

GeeksforGeeks. (2024). What is Proof of Concept (POC) in Software Development? [Web page]. Retrieved from: https://www.geeksforgeeks.org/what-is-proof-of-concept-poc-in-software-development/.

Genway, S., Neelakantaiah, G. K., Cruise, J. & Roberts, M. (2022). Avoiding quantum "POCitis": Five best practices life sciences organizations can embrace at the outset of every quantum computing proof of concept [Web page]. Retrieved from: https://www.capgemini.com/se-en/insights/expert-perspectives/avoiding-quantum-pocitis-five-best-practices-life-sciences-organizations-can-embrace-at-the-outset-of-every-quantum-computing-proof-of-concept-acr1/.

Gough, J., Bryant, D. & Auburn, M. (2021). *Mastering API Architecture*. O'Reilly Media. Retrieved from: https://books.google.ca/books?id=dMSVEAAAQBAJ.

Helgeson, J. (2009). *The Software Audit Guide*. ASQ Quality Press. Retrieved from: https://books.google.ca/books?id=9OqiEAAAQBAJ.

Hinchey, M. (2018). *Software Technology: 10 Years of Innovation in IEEE Computer*. Wiley. Retrieved from: https://books.google.ca/books?id=uo9jDwAAQBAJ.

Ingeno, J. (2018). *Software Architect's Handbook: Become a successful software architect by implementing effective architecture concepts*. Packt Publishing. Retrieved from: https://books.google.ca/books?id=6EZsDwAAQBAJ.

Invensis Learning. (2024). Proof of Concept (POC) [Web page]. Retrieved from: https://www.invensislearning.com/blog/proof-of-concept-poc/.

Jobin, C., Hooge, S. & Le Masson, P. (2020, Jun). What does the proof-of-concept (POC) really prove? A historical perspective and a cross-domain analytical study. *XXIXème conférence de l'Association Internationale de Management Stratégique (AIMS)*. Retrieved from: https://hal.science/hal-02570321.

Kupitman, Y. (2019). How Proof-of-Concepts Improve the Innovation Process [Blog post]. Retrieved from: https://proov.io/blog/how-proof-of-concepts-improve-the-innovation-process/.

Larman, C. (2002). *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and the Unified Process*. Prentice Hall PTR. Retrieved from: https://books.google.ca/books?id=r8i-4En_aa4C.

Lindsey, M. (2023). Defender EASM: Performing a Successful Proof of Concept (PoC) [Blog post]. Retrieved from: https://techcommunity.microsoft.com/t5/microsoft-defender-external/defender-easm-performing-a-successful-proof-of-concept-poc/ba-p/3994862.

McDowall, J. (2019). *Complex Enterprise Architecture: A New Adaptive Systems Approach*. Apress. Retrieved from: https://books.google.ca/books?id=QvmGDwAAQBAJ.

McGraw, G. & Felten, E. (1997). *Java Security*. Wiley. Retrieved from: https://books.google.ca/books?id=WIo_AQAAIAAJ.

Microfilms, U., Microfilms, X. U. & International, U. M. (1998). *Dissertation Abstracts International: The sciences and engineering. B*. University Microfilms. Retrieved from: https://books.google.ca/books?id=JI4vAQAAIAAJ.

Mistrik, I. (2012). *Aligning Enterprise, System, and Software Architectures*. Business Science Reference. Retrieved from: https://books.google.ca/books?id=lv3EswzXZpsC.

Morris, K. (2016). *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media. Retrieved from: https://books.google.ca/books?id=BIhRDAAAQBAJ.

Mun, C. L. (1965). Polytechnic education. *Toward a better future: Education and training for economic development in Singapore since*, 135–148.

National Archives and Records Administration. (2019). Guidance for a Proof of Concept Pilot for Electronic Records Management (ERM) [Web page]. Retrieved from: https://www.archives.gov/records-mgmt/policy/pilot-guidance.html.

Netguru. (2021). Proof of Concept in software development [Web page]. Retrieved from: https://www.netguru.com/blog/proof-of-concept-in-software-development.

New York Writing Publishing Center. (2023). What is a Proof of Concept (POC)? A Comprehensive Writing Guide + Free Template [Web page]. Retrieved from: https://www.linkedin.com/pulse/what-proof-concept-poc-comprehensive-writing-guide-free-nywpc/.

of Electrical, I. & Engineers, E. (1996). *Computer & Control Abstracts*. Institution of Electrical Engineers. Retrieved from: https://books.google.ca/books?id=iCYvAAAAMAAJ.

of New York at Binghamton. Department of Geography, S. U., of Geography, K. S. U. D. & of Geography, R. P. I. D. (1987). *Papers and Proceedings of Applied Geography Conferences*. Retrieved from: https://books.google.ca/books?id=-e6AAAAAIAAJ.

on Governmental Affairs, U. S. C. S. C. (1993). *IRS Computer Modernization and Procurement: Hearings Before the Committee on Governmental Affairs, United States Senate, One Hundred Second Congress, First and Second Session, June 25, 1991 and April 2, 1992*. U.S. Government Printing Office. Retrieved from: https://books.google.ca/books?id=gDR1qiYVFBoC.

on Governmental Affairs. Subcommittee on Oversight of Government Management, U. S. C. S. C. & the District of Columbia. (1995). *Reducing the Cost of Pentagon Travel Processing: Hearing Before the Subcommittee on Oversight of Government Management and the District of Columbia of the Committee on Governmental Affairs, United States Senate, One Hundred Fourth Congress, First Session, March 28, 1995*. U.S. Government Printing Office. Retrieved from: https://books.google.ca/books?id=gNtcsmL2fUUC.

on Science Space, C. & Technology. (1994). *Establishment of a Design Council in the Department of Commerce: Hearing Before the Subcommittee on Technology, Environment, and Aviation of the Committee on Science, Space, and Technology, U.S. House of Representatives, One Hundred Third Congress, Second Session, June 21, 1994*. U.S. Government Printing Office. Retrieved from: https://books.google.ca/books?id=z39ouIM0XYQC.

on Science. Subcommittee on Energy, U. S. C. H. C. & Environment. (1994). *High Performance Computing and Communications: Hearing Before the Subcommittee on Science of the Committee on Science, Space, and Technology, U.S. House of Representatives, One Hundred Third Congress, Second Session, May 10, 1994*. U.S. Government Printing Office. Retrieved from: https://books.google.ca/books?id=24BkjCB_eoMC.

on Science. Subcommittee on Energy, U. S. C. H. C. & Environment. (1996). *The Department of Energy's FY 1997 Budget Request for the Office of Energy Research (OER): Hearing Before the Subcommittee on Energy and Environment of the Committee on Science, U.S. House of Representatives, One Hundred Fourth Congress, Second Session, May 8, 1996*. U.S. Government Printing Office. Retrieved from: https://books.google.ca/books?id=XoEBJvWtD14C.

Orient Software. (2023). POC in IT: What it is, its importance, and tips for building an effective one [Web page]. Retrieved from: https://www.orientsoftware.com/blog/poc-in-it/.

Oxford. [Oxford Advanced Learner's Dictionary, 5th edition]. (2024). Definition of proof of concept noun. Oxford University Press. Retrieved on 2024-03-14 from: https://www.oxfordlearnersdictionaries.com/definition/english/proof-of-concept.

Pearson, T. (2024). Why Resource Misallocation Derails Software Project Teams [Blog post]. Retrieved from: https://spin.atomicobject.com/resource-misallocation-software/.

Perera, S. (2023). *Software Architecture and Decision-Making: Leveraging Leadership, Technology, and Product Management to Build Great Products*. Pearson Education. Retrieved from: https://books.google.ca/books?id=ph7YEAAAQBAJ.

Prateek, S. (2024). PoC vs MVP vs Prototype: Which Strategy to Use When? [Blog post]. Retrieved from: https://appinventiv.com/blog/poc-vs-mvp-prototype-the-best-strategy/.

Putra, A. [Anton Putra]. (2024, August, 17). Quarkus (Java) vs Fiber (Go): Performance Benchmark in Kubernetes 201 [Youtube Video]. Retrieved from: https://www.youtube.com/watch?v=PL0c-SvjSVg.

Recapped.io. (2022). 4 Tips for Running a Successful Proof of Concept [Web page]. Retrieved from: https://www.recapped.io/blog/4-tips-for-running-a-successful-proof-of-concept.

Rouse, M. (2024). proof of concept (POC) [Web page]. Retrieved from: https://www.techtarget.com/searchcio/definition/proof-of-concept-POC.

Rozenblit, J., Ewing, T., Schulz, S., on the Engineering of Computer Based Systems, I. C. S. T. C. & of Arizona, U. (1997). *Engineering of Computer-Based Systems, 1997 Conference*. IEEE Computer Society Press. Retrieved from: https://books.google.ca/books?id=J6I-AQAAIAAJ.

Sbarski, P., Cui, Y. & Nair, A. (2022). *Serverless Architectures on AWS, Second Edition*. Manning. Retrieved from: https://books.google.ca/books?id=g09gEAAAQBAJ.

Serra, J. (2024). *Deciphering Data Architectures*. O'Reilly Media. Retrieved from: https://books.google.ca/books?id=K0_zEAAAQBAJ.

Shrivastava, S., Srivastav, N., Sheth, R., Karmarkar, R. & Arora, K. (2022). *Solutions Architect's Handbook: Kick-start your career as a solutions architect by learning architecture design principles and strategies*. Packt Publishing. Retrieved from: https://books.google.ca/books?id=zTFYEAAAQBAJ.

Society, I. C. (1991). *Proceedings of the ... International IEEE Conference on Tools for Artificial Intelligence*. IEEE Computer Society Press. Retrieved from: https://books.google.ca/books?id=RvhVAAAAMAAJ.

Team, N. A. W. C. U. T. P. (1996). *Technology Transfer Today: Gaining the Competitive Edge*. Technical Publishing Team, Naval Air Warfare Center, Aircraft Division. Retrieved from: https://books.google.ca/books?id=1h7bKm8V-SkC.

Team, P. (1997). *CFO: Architect of the Corporation's Future*. Wiley. Retrieved from: https://books.google.ca/books?id=N4cpAQAAMAAJ.

TechBohdana, M. (2024). PoC vs Prototype vs MVP: A Detailed Comparison [Blog post]. Retrieved from: https://www.techmagic.co/blog/poc-vs-prototype-vs-mvp/.

TechnologyAdvice. (2023). Proof of Concept: A Complete Guide [Web page]. Retrieved from: https://technologyadvice.com/blog/project-management/proof-of-concept/#:~:text=Deliverables%3A%20List%20the%20expected%20outcomes,the%20end%20/.

TechTarget Contributor. (2024). Free proof of concept templates for the CIO [Web page]. Retrieved from: https://www.techtarget.com/searchcio/tip/Free-proof-of-concept-templates-for-the-CIO.

Tune, N. & Perrin, J. (2024). *Architecture Modernization: Socio-technical alignment of software, strategy, and structure*. Manning. Retrieved from: https://books.google.ca/books?id=MrvuEAAAQBAJ.

Van Hecke, W. (2013). *Learning iOS Design: A Hands-On Guide for Programmers and Designers*. Pearson Education. Retrieved from: https://books.google.ca/books?id=PkNGSW-2i-EC.

Wasonga, S. (2023). Performing a Successful MDTI Proof of Concept (PoC) [Blog post]. Retrieved from: https://techcommunity.microsoft.com/t5/microsoft-defender-threat/performing-a-successful-proof-of-concept-poc/ba-p/3742412.

Welker, J. E. (1963). SEASAT altimetry for surface height of inland seas. *NASA Technical Memorandum*.

Woods, E., Erder, M. & Pureur, P. (2021). *Continuous Architecture in Practice: Software Architecture in the Age of Agility and DevOps*. Pearson Education. Retrieved from: https://books.google.ca/books?id=itnPEAAAQBAJ.

Yeung, T. & Bull, A. (2022). Common Challenges with Conducting an Edge AI Proof of Concept [Blog post]. Retrieved from: https://developer.nvidia.com/blog/common-challenges-with-conducting-an-edge-ai-proof-of-concept/.