UNIVERSITY OF SÃO PAULO
INSTITUTE OF MATHEMATICS AND STATISTICS
BACHELOR OF COMPUTER SCIENCE

ArchHypo Tooling

*Enabling Practical Hypothesis Engineering
for Software Architecture*

Pedro Henrique Mariano Corrêa

FINAL ESSAY

MAC 499 — CAPSTONE PROJECT

Supervisor: José Gonçalves Lima Neto
Co-supervisor: Prof. Dr. Paulo Roberto Miranda Meirelles

Esta seção é opcional e fica numa página separada;
ela pode ser usada para uma dedicatória ou epígrafe.

Agradecimentos

Do. Or do not. There is no try.
— Mestre Yoda

Agradecer.

Abstract

Pedro Henrique Mariano Corrêa. **ArchHypo Tooling:** *Enabling Practical Hypothesis Engineering for Software Architecture*. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo.

Abstract text content.

Keywords: Keyword1. Keyword2. Keyword3.

Contents

Chapter 1

Introduction

1.1 Software Architecture and the Challenge of Rapid Evolution

In the contemporary landscape of software engineering, the definition and evolution of Software Architecture (SA) have transcended the traditional view of static blueprints to become a dynamic and continuous process essential for business survival. As software systems grow in complexity and scale, the architectural foundation upon which they are built dictates not only their current operational capacity but, more crucially, their future viability. Consequently, "Software architecture (SA) is a fundamental concept in software development, forming the structural foundation of systems and ensuring their long-term sustainability." This sustainability is not merely a technical concern; it is a strategic asset that determines an organization's agility in response to market fluctuations.

Historically, the academic understanding of architecture has been grounded in the tripartite definition proposed by Perry and Wolf, establishing SA as a composition of "three critical components: elements, form, and rationale." The *elements* represent the processing, data, and connecting components; the *form* dictates the weighted properties and relationships among them; and the *rationale* provides the justification for these structural choices. However, in the modern era of software development—characterized by the proliferation of startups and the adoption of Agile methodologies—the *rationale* component faces unprecedented pressure. The reasoning behind architectural decisions is often obfuscated by the urgency of delivery, leading to systems where the "form" degrades into what is colloquially known as a "Big Ball of Mud."

The core tension in modern software engineering lies in the temporal dimension of design. In static environments, architecture could be heavily front-loaded; however, "maintaining architectural integrity while enabling rapid feature delivery is crucial in fast-paced environments such as software startups, where requirements emerge frequently." This volatility in requirements necessitates a paradigm shift from architecture as a fixed deliverable to architecture as a continuous flow of decisions. The ability to pivot, scale, or refactor without catastrophic system failure is the hallmark of a robust architecture.

Therefore, the overarching challenge for practitioners and researchers alike resides in the delicate equilibrium between velocity and stability. The modern challenge resides in balancing "speed with quality to ensure sustainable, adaptable architectures that meet evolving business demands." Failure to maintain this balance results in technical debt accumulation that eventually paralyzes the development process, rendering the system incapable of evolving alongside the business it supports.

1.2 The Central Paradox: Decentralization vs. Tacit Knowledge

Despite the widespread recognition of the need for architectural flexibility, a profound dissonance exists between the organizational aspirations of the software industry and the methodological realities of architectural design. The industry, driven by the philosophy of DevOps and Team Topologies, increasingly advocates for the democratization of decision-making. The goal is to empower autonomous, cross-functional teams to own their architectural choices to eliminate bottlenecks and accelerate the flow of value. However, this aspiration creates a significant friction point when contrasted with the prevailing methods for architectural derivation.

This study identifies and addresses the "Paradox of Software Architecture Decision-Making," a phenomenon that "highlights the conflicting dynamics between industry reports advocating for decentralized, team-driven architectural decision-making and academic evidence showing that methods for deriving software architecture still heavily rely on the tacit knowledge of experienced practitioners."

This paradox is not merely anecdotal but is supported by rigorous secondary studies. A systematic mapping of the literature has revealed that existing methodologies for bridging the gap between requirements and architecture are fundamentally flawed in their reliance on seniority. The mapping revealed that existing methods "strongly relied on experienced practitioners' tacit knowledge to derive the architectural definitions for a software system." This reliance assumes the presence of a seasoned architect capable of intuitively synthesizing complex requirements into coherent structures—a resource that is scarce, expensive, and often unavailable in leaner startup environments.

The implications of this dependency are severe for organizations attempting to scale. When architectural methods are opaque and rooted in the intuition of a few experts rather than explicit, teachable processes, it "creates bottlenecks and limits the practical applicability of these methods in teams without senior members." It effectively concentrates power and responsibility in a centralized manner, directly contradicting the agile ethos of distributed ownership.

Consequently, a vicious cycle emerges. Organizations strive for decentralization to handle the fast flow of requirements, yet the tools and methods available to them force a reversion to centralized command-and-control structures to ensure architectural coherence. The result is a self-reinforcing cycle where the change to decentralization is "obstructed by frameworks that inherently require centralization, blocking the practical adoption of a socio-technical architecture approach." Breaking this cycle requires a new methodological

approach that externalizes tacit knowledge and makes the management of architectural uncertainty accessible to the broader engineering team.

1.3   The Philosophical Solution: ArchHypo

To resolve the paradox of centralized knowledge in a decentralized world, this work adopts and expands upon the principles of **ArchHypo**. This technique represents a philosophical shift from viewing architecture as a set of facts to viewing architecture as a set of experiments. By acknowledging that early-stage architectural decisions are often based on assumptions rather than certainties, ArchHypo "uses hypotheses engineering to manage uncertainties related to software architecture."

The core philosophy of ArchHypo aligns seamlessly with the scientific method. Rather than making premature commitments to specific technologies or patterns based on incomplete information—a practice that leads to rigid and brittle systems—ArchHypo advocates for the explicit documentation of uncertainties. The objective is to allow the "strategic postponement of decisions while addressing their potential impact." This is not procrastination; it is a deliberate risk management strategy. By treating an architectural choice as a hypothesis (e.g., "The application should handle 1000 simultaneous requests"), the team can define metrics and experiments to validate that choice before fully committing to it.

This approach provides a structured mechanism for the implementation of the "Continuous Architecture" principle to "delay design decisions." In traditional agile environments, delaying a decision is often conflated with ignoring it. ArchHypo, conversely, ensures that the delayed decision is actively managed. It provides a framework that supports the team in "evaluating the respective trade-offs in identifying the most responsible moment for that decision." The "Responsible Moment" is the point in time where the cost of deciding is outweighed by the cost of waiting, and where sufficient information has been gathered to minimize risk.

Furthermore, ArchHypo moves beyond mere identification of uncertainty to actionable engineering. It proposes "formulating a technical plan based on each hypothesis' assessment, incorporating measures able to mitigate its impact and reduce uncertainty." This might involve creating architectural spikes, implementing tracer bullets, or designing abstraction layers that isolate the uncertain component. Through this philosophical lens, architecture becomes an iterative, evidence-based process that can be safely distributed among team members, as the decision-making criteria are explicit rather than hidden within an expert's intuition.

1.4   The Practical Contribution: From Theory to Tooling

While ArchHypo offers a robust theoretical framework for managing architectural uncertainty, its practical application in real-world scenarios has encountered significant friction. Theoretical elegance does not always translate to operational efficiency without the appropriate support mechanisms. Empirical research conducted to evaluate ArchHypo

in industrial settings, although demonstrating benefits such as a structured approach to architectural work, revealed "significant adoption challenges."

The cognitive load required to shift from a feature-centric mindset to a hypothesis-centric mindset is substantial. In observational studies, the research team identified the "learning curve and process adjustments required for ArchHypo's adoption as significant challenges that could hinder its widespread adoption." Engineering teams, often under pressure to deliver functional requirements, found the manual management of hypotheses to be administratively burdensome. Specifically, participants found the technique "hard to learn, particularly in mapping risks, specifying hypotheses, and defining action plans." Without guidance, the specification of hypotheses was inconsistent, and the link between a hypothesis and its technical mitigation plan was often tenuous.

These findings underscored a critical gap in the current state of the art: there is "a strong need for better guidance and, crucially, tool support to manage hypotheses and execute action plans, which could lessen team dependency on architects." If the goal is to democratize architectural decision-making, the method must be supported by tooling that guides less experienced developers through the process, reducing the reliance on the tacit knowledge of senior architects.

Therefore, the primary practical contribution of this capstone is the design and development of **HypoStage**. Recognizing the industry trend towards Internal Developer Portals (IDPs) as central hubs for engineering operations, HypoStage is defined as an "ArchHypo plugin for Backstage." By integrating directly into the developer's existing workflow within the Backstage ecosystem, HypoStage aims to lower the barrier to entry for Hypothesis Engineering.

HypoStage provides a comprehensive digital structure that operationalizes the theoretical concepts of ArchHypo. It includes features for "Uncertainty Assessment, Quality Attributes Tracking, and Technical Planning capabilities," allowing teams to document hypotheses, score them based on impact and uncertainty, and track the execution of experimental mitigation plans. Through this tool, this work aims to bridge the gap between the theoretical promise of decentralized decision-making and the practical reality of fast-paced software development, finally resolving the paradox of architectural decision-making.

Chapter 2

Theoretical and Architectural Basis

This chapter establishes the theoretical foundations necessary for the development of the proposed solution. It begins by contextualizing Software Architecture within dynamic and agile environments, identifying the friction between rapid delivery and architectural integrity. Subsequently, it delineates the central problem motivating this work: the Paradox of Software Architecture Decision-Making. To address this paradox, the chapter introduces the concepts of Hypothesis Engineering and the ArchHypo framework, detailing its processes, assessment metrics, and the pattern language that supports technical planning. Finally, the chapter presents the practical gap identified in empirical studies, justifying the necessity for the development of specific tooling to support these practices.

2.1 The Role of Software Architecture in Agile and High-Flow Environments

Software Architecture (SA) is a fundamental concept in software development, forming the structural foundation of systems and ensuring their long-term sustainability. While early definitions, such as those by Perry and Wolf in 1992, characterized SA as a composition of elements, form, and rationale, modern definitions have evolved to incorporate deeper insights into the dynamics of system evolution. Contemporary definitions, such as those by Richards and Ford, characterize SA as a high-level system structure encompassing essential characteristics, design principles, and the decisions to guide system adaptability and maintainability.

This expanded perspective highlights that SA is not merely a static blueprint but a dynamic framework that guides decisions throughout the system life cycle and aligns technical and business needs. However, the practical application of architectural design is fraught with complexity, particularly regarding the timing and certainty of decisions.

2.1.1 The Challenge of Architecture in Agile Contexts

Agile methodologies have successfully introduced practices like Test-Driven Development (TDD) and refactoring, yet they often lack well-recognized and agreed-upon

approaches for architectural design. In many agile contexts, the most recognized approach is to simply let the architecture "emerge" and refine it over the life of the project. While this approach aims to reduce upfront design, it often leads to a lack of established practice, meaning architectural changes remain challenging within agile contexts, often necessitating significant upfront design that contradicts core agile principles.

In rapidly evolving business environments, such as software startups, new requirements continually reshape architectural demands. In these dynamic scenarios, architecture anti-patterns—such as the "Big Ball of Mud"—may emerge, posing risks to effective architectural evolution and resulting in systems that struggle to maintain proper modularity and scalability. This anti-pattern often emerges when immediate demands and incremental fixes override deliberate architectural planning. Consequently, engineering teams must balance speed with quality to ensure sustainable, adaptable architectures that meet evolving business demands.

2.1.2 Trends in Evolutionary Architecture

To address the friction between agility and structure, concepts such as *Agile Architecture* and *Continuous Architecture* have emerged. These frameworks emphasize principles such as delaying design decisions and architecting for change. The core philosophy is that decisions can be postponed until they are absolutely necessary, ensuring that architectures are based on facts rather than guesses.

However, determining which decisions to delay or where flexibility is beneficial incurs significant challenges. Time emerges as a critical dimension in architectural design, with iterative planning being essential, particularly for solutions undergoing continuous evolution.

2.1.3 The Necessity of Fast Flow

Recent industry trends emphasize the concept of "fast flow," which refers to an organization's ability to continuously and rapidly deliver software changes that align with evolving business needs while preserving the system health and architectural integrity. Achieving fast flow relies on autonomous, empowered teams supported by self-service platforms that minimize operational blockers.

This concept is intrinsically linked to the adoption of socio-technical architecture, which reflects the growing realization that architectural decisions should not be confined to a select few architects but distributed across development teams. The socio-technical approach promotes team autonomy and empowerment, advocating for decision-making processes that include all team members, regardless of their experience levels.

2.2 The Paradox of Software Architecture Decision-Making

Despite the industry's aspiration for decentralized, autonomous teams capable of maintaining a "fast flow" of requirements, a significant theoretical and practical barrier

exists. This research identifies this barrier as the **Paradox of Software Architecture Decision-Making**.

2.2.1 The Central Paradox

The Paradox of Software Architecture Decision-Making highlights the conflicting dynamics between industry reports advocating for decentralized, team-driven architectural decisions to enable a fast flow of requirements, and academic research indicating that existing methods for deriving SA still heavily depend on the tacit knowledge of experienced practitioners.

While the industry moves toward distributed architectural ownership, the current methods and frameworks continue to reinforce centralized decision-making. This paradox creates a self-reinforcing cycle: while organizations may aim for a decentralized approach to support a fast flow of requirements, they remain dependent on expert-driven methods that, by design, limit the involvement of less experienced team members.

2.2.2 Reliance on Tacit Knowledge

A systematic mapping study by Souza et al. (2019) examined methods and practices for deriving architectural models from requirements specifications. A key finding was that existing methods strongly relied on experienced practitioners' tacit knowledge to derive the architectural definitions for a software system. This reliance on the intuition and expertise of architects makes it difficult for teams without such expertise to adopt these methods effectively.

This dependency creates bottlenecks that limit an effective decentralization of decision-making. As a result, the shift towards a more distributed decision-making model is obstructed by frameworks that inherently require centralization, blocking the practical adoption of a socio-technical architecture approach.

2.2.3 Deficiencies in Existing Methods

Beyond the reliance on tacit knowledge, the systematic mapping study highlighted several other gaps in existing architectural derivation methods:

- **Lack of Tool Support:** About a third of analyzed architectural derivation approaches lack tool support. The study noted that only 30.7% of methods provided decision-making support.

- **Inadequate Support for Non-Functional Requirements (NFRs):** Although NFRs such as performance, scalability, and security are essential, many approaches focus primarily on functional requirements, neglecting NFRs in architectural decision-making processes.

- **Limited Empirical Validation:** Many methods have not been sufficiently validated in real-world, dynamic environments. Over half lack explicit evaluation methods.

This points to a clear need for tools to aid architects, suggesting patterns and supporting decision-making to reduce dependency on seasoned experts.

2.3 Hypothesis Engineering and Uncertainty Management

To resolve the paradox of decision-making and enable decentralized architecture, it is necessary to move away from reliance on tacit intuition toward a more explicit management of the unknowns. This is achieved through **Hypothesis Engineering**.

2.3.1 The Concept of Hypothesis Engineering

Hypothesis Engineering offers a philosophical approach to managing uncertainty by treating assumptions as testable hypotheses. It is defined as a process of continuously validating product assumptions, transforming them into hypotheses, prioritizing and testing them following the scientific method to support or refute them.

In this context, the word "hypothesis" focuses on business assumptions that should be evaluated before start-ups develop their business models. However, this concept is extended here to manage the uncertainty of architectural decisions.

2.3.2 Distinguishing Uncertainty from Risk

A critical theoretical distinction must be made between **risk** and **uncertainty**. Risk involves evaluating the probability and impact of specific events; it represents the likelihood of an event occurring and its potential consequences.

In contrast, **uncertainty** refers to the lack of information necessary for architectural decisions. Uncertainty represents the lack of information to make a decision; in cases where partial information is available, the uncertainty is related to what is still unknown. ArchHypo is based on the premise that uncertainties related to the software architecture are natural in all stages of a software project and that, instead of resisting them, a better approach would be to embrace and manage them.

2.3.3 Characteristics of an Architectural Hypothesis

An architectural hypothesis represents any uncertain statement relevant to the software architecture design. Its primary characteristic is that it must be **falsifiable**—that is, the possibility of proving it false exists.

Unlike a requirement statement, which is usually assumed as true when written, using the word "hypothesis" makes it clear to the whole team that it represents uncertainty. Hypotheses must be shared with the development team so that everyone is aware of the uncertainties that can affect architectural decisions. This explicit documentation allows the team to systematically test and validate assumptions, gather data, and adjust their approach as needed.

2.4 The ArchHypo Framework

ArchHypo is a technique that uses hypothesis engineering to manage uncertainties related to software architecture and enhance decision-making processes. It provides a structured framework for software architects and developers, enabling them to make uncertainties explicit and manageable.

2.4.1 Sources of Uncertainty

When applying ArchHypo, a team focuses specifically on hypotheses that can affect the software architecture. The most common sources of uncertainty are in the **requirements** and the **solutions**.

- **Requirements Uncertainty:** This occurs when a requirement is based on limited evidence or lacks important details. For instance, uncertainty regarding the number of simultaneous requests a system must handle.

- **Solution Uncertainty:** This relates to the unknown consequences of current architecture components or candidate solutions. For example, uncertainty regarding whether a specific library is compatible with a required protocol.

2.4.2 Assessment: Uncertainty Level and Impact

Once a hypothesis is identified, it is assessed based on two independent dimensions:

- **Uncertainty Level:** This reflects how far the team is from proving that a hypothesis is true or false. The uncertainty would be high if there is a lack of information to estimate probability or when alternatives have a similar chance to happen.

- **Impact:** This measures the effort required to transition to a different alternative. It represents the consequences and costs that the uncertainty can cause.

This assessment is typically performed qualitatively by the team, using a scale (e.g., a five-point Likert scale: Very Low to Very High). These assessments aim to provoke reflection about the hypotheses, allowing a qualitative comparison among them and helping in choosing techniques for handling the respective uncertainty.

2.4.3 The Technical Plan

The core operational mechanism of ArchHypo is the formulation of a **Technical Plan** based on each hypothesis' assessment. This plan is derived from the pattern *Plan for Responsible Moments.*

The plan can include actions aiming to:

- Definitely accept or refute the hypothesis.

- **Reduce the uncertainty:** Giving the team more confidence to move forward with a decision.

- **Reduce the impact:** Allowing the team to "stay longer with the uncertainty" by isolating the affected areas or increasing flexibility.

- Define criteria (triggers) to postpone handling the hypothesis until a specific condition is met.

2.4.4   Supporting Practices and Patterns

The technical plan utilizes a specific pattern language to address uncertainties. Key patterns include:

- **Architectural Spike:** Small technical experiments in which working software is created to prove or disprove the feasibility of a specific hypothesis.

- **Tracer Bullet:** An architecturally significant functionality that helps to exercise and demonstrate an end-to-end path inside the architecture, aiming to evaluate how new technologies could be integrated. This pattern provides a concrete way to design the basic application architecture.

- **Software Analytics:** Identifying relevant metrics to collect and analyze to assess and monitor a given quality attribute of the system.

- **Architectural Trigger:** Defines conditions that trigger architectural investigations which may lead to adding tasks to the backlog. For example, having a given number of simultaneous accesses might fire a trigger related to scalability.

- **Development Guidelines:** Small adjustments in the development process to deal with recurrent uncertainties. These include:

- **Protective Guideline:** Defining programming practices to be followed or avoided to not limit options for an architectural decision being postponed.

- **Bring the Specialist:** Involving individuals with the right skills or knowledge in activities where this expertise can reduce the uncertainty.

- **Plan for Preparation:** Introducing steps to obtain information before activities that recurrently have an associated uncertainty.

- **Quality Checkpoint:** Introducing a verification activity after the development of an artifact to verify if the desired quality is present.

2.5   The Practical Gap and the Need for Tooling

While the theoretical framework of ArchHypo is robust, empirical evidence suggests that its manual application presents significant challenges that hinder widespread adoption.

2.5.1   Adoption Barriers

An empirical study on ArchHypo reported that while the technique provided a structured approach to dividing architectural work, the team identified the learning curve and process adjustments required for adoption as significant challenges. Specifically,

participants found the technique hard to learn, particularly in mapping risks, specifying hypotheses, and defining action plans.

Team members mentioned difficulty in "mapping the scenarios with risk" to the hypothesis and "definition of the actions" to handle the hypothesis. One participant explicitly noted that "train[ing] the team a little more on how to find, map risk scenarios, and define the necessary actions... would make the team less dependent on the team of architects".

2.5.2   The Necessity of Tool Support

The identified difficulties highlight a strong need for better guidance and, crucially, **tool support to manage hypotheses and execute action plans**, which could lessen team dependency on architects. The current literature notes that about a third of analyzed architectural derivation approaches lack tool support.

Tools that provide more guidance on using the technique can be developed to reduce its learning curve. Without effective tools to aid architects, suggesting patterns and supporting decision-making, the dependency on seasoned experts remains a bottleneck.

2.5.3   Integration with Developer Platforms

To bridge this gap, researchers have suggested that plugins of existing tools could introduce support for hypotheses management. By integrating with project management platforms or developer portals, a tool can facilitate seamless adoption into diverse development workflows.

The development of **HypoStage**, a dedicated software tool, aims to operationalize the ArchHypo framework. It intends to allow teams to explicitly document and manage architectural hypotheses, assess them based on impact and uncertainty, and define and track technical action plans. This technological intervention addresses the core of the "Paradox of Software Architecture Decision-Making" by providing the structural scaffolding necessary for less experienced practitioners to engage in decentralized, high-quality architectural decision-making.

Chapter 3

The HypoStage Tooling Solution

This chapter presents the practical contribution of this research: the conception, architectural design, and implementation of **HypoStage**. Based on the theoretical foundations established in the previous chapter and the empirical gaps identified in the literature regarding the ArchHypo framework, this chapter details the development of a software tool designed to operationalize decentralized architectural decision-making. The chapter begins by justifying the necessity of a technological intervention to resolve the "Paradox of Software Architecture Decision-Making." It then proceeds to describe the solution's integration within the Backstage internal developer portal, details the functional requirements for managing the hypothesis lifecycle, and concludes with a technical examination of the software architecture and the open-source strategy adopted for the project.

3.1   Justification for Tooling Support

The theoretical framework of ArchHypo, as discussed in Chapter 2, provides a robust methodology for managing architectural uncertainty through Hypothesis Engineering. However, the transition from theory to practice presents significant hurdles. Empirical research conducted by Silva et al. on the application of ArchHypo has demonstrated clear benefits, such as improved collaboration and a structured approach to architectural work, yet it simultaneously revealed "significant adoption challenges".

3.1.1   The Dilemma of Manual Application

The primary barrier to the widespread adoption of Hypothesis Engineering in software architecture lies in the cognitive load and process overhead imposed on development teams. Participants in empirical studies found the technique "hard to learn, particularly in mapping risks, specifying hypotheses, and defining action plans". Specifically, the abstract nature of translating vague architectural uncertainties into testable hypotheses proved to be a cognitive bottleneck. The study highlighted that teams struggled with "mapping the scenarios with risk" and properly articulating the necessary technical interventions.

Furthermore, without structured guidance, the execution of the framework often

defaulted back to the most experienced members of the team, reinforcing the very centralization the method aims to dismantle. The feedback from practitioners was explicit regarding the solution to this problem: there is a strong need for "tool support to manage hypotheses and execute action plans, which could lessen team dependency on architects". Without effective tools to aid architects—suggesting patterns and supporting decision-making—the dependency on seasoned experts remains a bottleneck, perpetuating the centralization of architectural authority.

3.1.2 Technological Intervention as a Scaffolding Mechanism

This capstone project posits that the "Paradox of Software Architecture Decision-Making"—where the desire for decentralized autonomy conflicts with the reliance on centralized tacit knowledge—cannot be resolved through process changes alone. It requires a technological intervention. The development of a dedicated software tool aims to "operationalize the ArchHypo framework".

By embedding the methodological rules, assessment scales, and technical patterns directly into a software interface, the tool provides the necessary structural scaffolding for less experienced practitioners. It intends to allow teams to "explicitly document and manage architectural hypotheses, assess them based on impact and uncertainty, and define and track technical action plans". Consequently, the tool acts as a mechanism for knowledge distribution, enabling the decentralization of high-quality architectural decision-making.

3.2 HypoStage and the Backstage Platform Strategy

To address the identified needs, this project introduces **HypoStage**, a software solution designed to integrate architectural hypothesis management into the daily workflow of engineering teams.

3.2.1 The HypoStage Solution

HypoStage is defined as an "ArchHypo plugin for Backstage". It is not merely a documentation repository but an active management tool that "integrates architectural hypothesis management into your Backstage environment, enabling teams to document, track, and validate architectural decisions effectively". The tool is designed to guide users through the lifecycle of a hypothesis, from elicitation to validation or refutation, ensuring that uncertainty is treated as a first-class citizen in the software delivery process.

A key design principle of HypoStage is its capability to "seamlessly integrates with Backstage's catalog and entity system". This ensures that architectural hypotheses are not abstract entities but are directly linked to the software components, APIs, and resources they affect, promoting traceability and context-aware decision-making.

3.2.2 Rationale for the Backstage Ecosystem

The decision to implement HypoStage as a plugin for **Backstage**—an open-source Internal Developer Portal (IDP)—is strategic. To bridge the gap identified in the literature,

researchers have suggested that "plugins of existing tools could introduce support for hypotheses management". By integrating with project management platforms or developer portals, a tool can "facilitate seamless adoption into diverse development workflows".

Backstage was selected as the host platform because it serves as a centralized hub for software development teams, aggregating infrastructure, services, and documentation. By placing hypothesis management within the IDP, HypoStage ensures that architectural decision-making occurs in the same environment where development happens. This visibility is crucial for decentralization, as it democratizes access to architectural rationale. Furthermore, the modular architecture of HypoStage allows it to be "integrated with existing developer portals... facilitating seamless adoption", leveraging Backstage's extensible plugin architecture to reach a wide audience of practitioners.

3.3 Functional Requirements and Implementation

HypoStage implements specific functional requirements derived from the ArchHypo framework to support the complete lifecycle of hypothesis engineering. These functionalities cover documentation, assessment, planning, and visualization.

3.3.1 Explicit Hypothesis Management and Tracking

The core functionality of HypoStage is "Hypothesis Management," which allows users to "create, edit, and track architectural hypotheses with detailed metadata". The system enforces a structured format for elicitation, requiring clear statements that define the uncertainty. The implementation provides a robust form interface that captures essential data points such as the "Hypothesis Statement," "Source Type" (e.g., Requirements, Solution), and associated "Entity References" from the catalog.

Furthermore, the tool supports "Status Tracking," allowing teams to "monitor hypothesis lifecycle from creation to validation". The system supports multiple statuses for hypotheses, such as "Open", "In Review", "Validated", "Discarded", and "Trigger-Fired". This explicit tracking ensures that architectural uncertainties are not forgotten but are actively managed until resolution.

3.3.2 Uncertainty and Impact Assessment

A critical component of the ArchHypo framework is the quantitative and qualitative assessment of hypotheses. HypoStage implements "Uncertainty Assessment" functionality to "evaluate hypothesis uncertainty using Likert scale ratings".

The interface requires users to provide an "Uncertainty Rating" and an "Impact Rating," both utilizing a "1-5 scale". These ratings map to values ranging from "Very Low" (1) to "Very High" (5). This structured assessment forces teams to critically evaluate how far they are from validating a hypothesis and what the potential consequences of failure are, directly addressing the difficulty participants faced in "mapping risks" in manual implementations.

3.3.3 Technical Planning and Quality Correlation

To move from assessment to action, HypoStage provides "Technical Planning" capabilities. The tool allows teams to "create and manage technical planning items linked to hypotheses". Users can define specific actions—such as "Architectural Spike," "Tracer Bullet," or "Prototype"—assigned to specific entities with target dates and expected outcomes. This directly supports the objective to "define and track technical action plans associated with each hypothesis".

Additionally, the tool implements "Quality Attributes Tracking," enabling users to "associate hypotheses with specific quality attributes". The system supports a comprehensive list of attributes, such as Performance, Security, Scalability, and Maintainability. This feature ensures that architectural decisions are explicitly linked to the non-functional requirements they impact, fostering a quality-driven architectural culture.

3.3.4 Visualization and Evidentiary Support

To support decision-making over time, HypoStage includes "Visualization" features to "track hypothesis evolution and validation status through interactive charts". The system renders a temporal view of how uncertainty and impact ratings change as technical plans are executed, providing empirical evidence of risk reduction.

Finally, to ground decisions in facts rather than intuition, the tool supports the inclusion of "Evidence URLs". The system allows teams to "add supporting documentation links", such as external test results, POC repositories, or vendor documentation, thereby ensuring that the validation of a hypothesis is auditable and transparent.

3.4 Software Engineering Architecture

The engineering of HypoStage follows modern software development practices, utilizing a decoupled architecture that aligns with the Backstage plugin ecosystem. The solution is divided into two main packages: the frontend plugin and the backend plugin.

3.4.1 Component Architecture

The system architecture is composed of distinct "frontend components" and "backend services".

Frontend Architecture

The frontend is built using React and integrates with the Backstage core API.

Backend Architecture

The backend logic is encapsulated in the backend package.

- HypothesisService: The "Core service for hypothesis management". As seen in this service manages business logic and data persistence. It utilizes database transactions

to ensure data integrity when creating hypotheses and logging associated lifecycle events simultaneously.

- Router: The router module uses express-promise-router to define RESTful endpoints, handling request validation via Zod schemas before invoking the service layer.

- Persistence: The system uses Knex.js for database interactions, including "Database migrations for schema setup" which define the structure for hypothesis, technicalPlanning, and hypothesisEvents tables.

3.4.2 Design for Integration and Modularity

The project adheres to the goal of ensuring "modularity and integrability". The plugin is registered within the Backstage backend system using the createBackendPlugin factory, allowing it to inject dependencies such as logger, database, httpAuth, and catalogService.

This dependency injection model is crucial for the tool's operation as a socio-technical enabler. For instance, the HypothesisService interacts with the CatalogService to fetch entity references, ensuring that hypotheses are tightly coupled to the actual software components registered in the organization's ecosystem. This design allows HypoStage to be "used as a standalone application" (in a dev environment) or "integrated with existing developer portals", fulfilling the requirement for seamless adoption.

3.5 Open Source Commitment and Documentation

Recognizing that the challenges of architectural decision-making are universal, HypoStage is positioned as a contribution to the wider software engineering community.

3.5.1 Licensing and Distribution

The project is distributed as open-source software under the LGPL-3.0 license (GNU Lesser General Public License v3.0). This licensing model was chosen to balance the freedom of use with the requirement that modifications to the plugin library itself remain open source, thereby encouraging community contribution back to the core tool while allowing integration into proprietary Backstage instances.

3.5.2 Documentation for Adoption

To mitigate the learning curve not just of the method, but of the tool itself, comprehensive documentation has been developed. The README.md serves as the primary entry point, detailing "Installation," "Configuration," and "Usage" guides. It explains how to configure the frontend routes and backend plugins, ensuring that organizations can adopt the tool with minimal friction. By providing clear "API Reference" and "Features" descriptions, the project aims to lower the barrier to entry, facilitating the shift from centralized, tacit architectural management to a transparent, decentralized, and hypothesis-driven approach.

Chapter 4

Use case

Chapter 5

Conclusion: Contributions and Future Directions

5.1 Summary of Contributions: The Artifact and the Paradox

The central investigation of this capstone project has been driven by a critical examination of the tension between the industry's demand for agility and the academic reality of architectural design methods. Through a systematic mapping of the literature and an analysis of grey literature, this work identified and defined the "Paradox of Software Architecture Decision-Making". This paradox encapsulates the conflict where the industry trend towards decentralized, autonomous teams—essential for maintaining a fast flow of requirements—collides with the methodological reality that current architectural derivation techniques "heavily rely on the tacit knowledge of experienced practitioners". This reliance creates a bottleneck, effectively reinforcing centralized decision-making structures and obstructing the practical adoption of socio-technical architecture approaches.

The theoretical resolution to this paradox lies in ArchHypo, a technique that shifts the architectural paradigm from static decision-making to dynamic hypothesis engineering. By framing architectural uncertainties as testable hypotheses, ArchHypo offers a structured pathway to manage risk and distribute decision-making power. However, theoretical frameworks alone are insufficient to change organizational behavior. Empirical evaluations of ArchHypo revealed that while the technique is effective, it presents a steep learning curve and administrative burden that can hinder widespread adoption. Consequently, this project posits that the Paradox of Software Architecture Decision-Making "cannot be resolved through process changes alone. It requires a technological intervention".

To address this critical gap, the primary contribution of this capstone project is the design and development of HypoStage, a specialized software artifact designed to operationalize the ArchHypo framework. Recognizing that modern engineering organizations are increasingly adopting Internal Developer Portals (IDPs) to centralize their operations, HypoStage is architected as an "ArchHypo plugin for Backstage". This strategic integra-

tion ensures that hypothesis management is not an extraneous activity but is embedded directly into the developer's daily workflow.

HypoStage serves as a "tangible solution to the identified gap in tool support for managing architectural uncertainty". It translates the abstract concepts of Hypothesis Engineering into a concrete digital interface, thereby reducing the cognitive load required to adopt the methodology. By automating the administrative aspects of the framework, the tool directly addresses the documented need for better guidance to "lessen team dependency on architects".

The fundamental objective of HypoStage is to "operationalize the ArchHypo framework". It achieves this by providing a structured environment that "integrates architectural hypothesis management into your Backstage environment, enabling teams to document, track, and validate architectural decisions effectively". This operationalization transforms ArchHypo from a theoretical concept into a pragmatic engineering practice.

Crucially, HypoStage acts as a mechanism for democratization. By embedding the methodological rules, assessment criteria, and workflow steps directly into the user interface, the tool provides the "structural scaffolding necessary for less experienced practitioners to engage in decentralized, high-quality architectural decision-making". This scaffolding externalizes the tacit knowledge that was previously the exclusive domain of senior architects, allowing broader team participation in the architectural process without compromising integrity.

In terms of specific capabilities, HypoStage empowers engineering teams to "Explicitly document and manage architectural hypotheses, assess them based on impact and uncertainty, and define and track technical action plans". The tool enforces rigor in the assessment process by utilizing "Likert scale ratings" for evaluating Impact and Uncertainty, ensuring that qualitative assessments are consistent and comparable across the organization. Through these features, HypoStage provides the technological foundation necessary to break the cycle of centralization and enable true socio-technical architectural evolution.

5.2 Results: Insights Empowered by the Tool

The development of HypoStage is grounded in the proven empirical benefits of the ArchHypo technique. The tool is designed not merely to record data, but to amplify and sustain the positive outcomes observed in industrial applications of the methodology. By facilitating the management of architectural hypotheses, the tool unlocks several critical advantages for software development projects.

Support for Prioritization and Risk Management

One of the most significant impacts of applying this methodology is the enhancement of risk visibility. In complex projects, architectural risks are often opaque or implicit. The structured elicitation and assessment of hypotheses supported by the tool allow for "a more clear visualization of the risks that could threaten the final delivery of the project". By making these risks explicit and quantifiable through impact and uncertainty scores, teams

can prioritize their efforts more effectively, focusing on the most critical uncertainties that could jeopardize project success.

Sustainable Task Distribution and Planning

The methodology provides a robust framework for managing the temporal dimension of architecture. Rather than treating architecture as a monolithic upfront phase, the approach supported by HypoStage enables a "structured approach to dividing the architectural work through iterations". This allows for "allocating and scheduling architectural tasks throughout project iterations", ensuring that architectural evolution is continuous and integrated with feature development. This distribution prevents the accumulation of unmanaged technical debt and ensures that architectural work is prioritized alongside functional requirements.

Sustainable Development and Quality

The systematic management of uncertainty contributes directly to the sustainability of the development process. Empirical evidence suggests that the adoption of this approach benefits projects by providing "predictability, security, transparency, quality, and sustainable pace". Furthermore, the technique has been shown to contribute significantly to "decision-making processes and process efficiency". By reducing the chaos associated with unforeseen architectural blockers, teams can maintain a steady velocity and deliver higher-quality software that meets both functional and non-functional requirements.

Reduction of Upfront Design

A core tenet of modern agile architecture is the avoidance of heavy upfront design (BDUF), which often leads to waste and rigidity. The ArchHypo technique, operationalized by the tool, allows projects to avoid "an upfront architectural design". It achieves this by providing a safety net that supports the "strategic postponement of decisions while addressing their potential impact". By identifying which decisions can be safely delayed and monitoring their associated risks, teams can maintain agility and keep their options open until the "Most Responsible Moment" arises.

Informed Decision-Making

Finally, the ultimate goal of the tool is to elevate the quality of architectural decisions. By moving away from intuition-based choices, the methodology ensures that decisions are grounded in evidence and analysis. Teams that have adopted this approach recognized that the management of hypotheses "makes the team base the decisions on information". HypoStage facilitates this by tracking the results of experiments, spikes, and analytics, ensuring that when a decision is finally made, it is backed by data rather than guesswork.

5.3 Future Steps: Validation and Research Evolution

While the development of HypoStage represents a significant step towards resolving the challenges of decentralized architectural decision-making, it marks the beginning of

a new phase of research and validation. The path forward involves rigorous empirical testing of the tool, the refinement of the underlying methodologies, and the continued expansion of the pattern language that supports ArchHypo.

Empirical Validation in Diverse Contexts

To fully understand the efficacy and generalizability of HypoStage, it is essential to conduct extensive empirical studies. Future work must "evaluate the adoption of ArchHypo in companies and development teams with different backgrounds and characteristics". This includes varying team sizes, domains, and maturity levels. Furthermore, researchers must "apply these patterns in other projects to study their broader impact, address challenges, and refine solutions to improve their adoption and effectiveness". These studies should aim to quantify the reduction in decision-making bottlenecks and the improvement in architectural quality.

User Acceptance Evaluation

The success of any developer tool depends on its acceptance by the practitioner community. To systematically evaluate how engineering teams perceive and interact with HypoStage, future interview and survey studies should be designed to identify the factors that influence the adoption of the tool—such as performance expectancy, effort expectancy, and social influence. Gathering this data will help refine both the user experience and the feature set.

Tool Improvement and Learning Curve Reduction

A primary motivation for HypoStage was to mitigate the steep learning curve associated with ArchHypo. Consequently, continuous investigation into "tools and methodologies to improve the implementation of ArchHypo represents an important direction for research, potentially expanding its applicability" is necessary. Future iterations of the tool should focus on features that provide "more guidance on using the technique can be developed to reduce its learning curve". This could include intelligent recommendation systems that suggest potential hypotheses based on project characteristics or automated generation of technical plans based on historical data.

Evolution of the Framework via New Patterns

The theoretical foundation of ArchHypo must also continue to evolve. Future research should focus on "identifying and documenting patterns for various types of hypotheses". As the software landscape changes, new categories of uncertainty emerge. There is a specific need for the "exploration of common uncertainties related to specific quality attributes, such as sustainability and usability". Documenting these patterns will enrich the knowledge base available to users of HypoStage, allowing them to leverage collective industry knowledge when formulating their own hypotheses.

Research on Process Guidelines

Finally, the integration of ArchHypo into the broader software development lifecycle remains a fertile area for research. The empirical study noted that the introduction of specific process guidelines was a highly effective strategy for managing uncertainty. Therefore, "future studies can also investigate how these guidelines can be adopted as an approach to dealing with uncertainty". Understanding how to best weave hypothesis engineering into Agile, DevOps, and other process methodologies will be crucial for the seamless adoption of decentralized architectural decision-making.