

ArchHypo: Managing Software Architecture Uncertainty Using Hypotheses Engineering

Kelson Silva , Jorge Melegati , Fabio Silveira , Xiaofeng Wang ,
Mauricio Ferreira , and Eduardo Guerra 

Abstract—Uncertainty is present in software architecture decisions due to a lack of knowledge about the requirements and the solutions involved. However, this uncertainty is usually not made explicit, and decisions can be made based on unproven premises or false assumptions. This paper focuses on a technique called ArchHypo that uses hypotheses engineering to manage uncertainties related to software architecture. It proposes formulating a technical plan based on each hypothesis’ assessment, incorporating measures able to mitigate its impact and reduce uncertainty. To evaluate the proposed technique, this paper reports an application of the technique in a mission-critical project that faced several technical challenges. Conclusions were based on data extracted from the project documentation and a questionnaire answered by all team members. As a result, the application of ArchHypo provided a structured approach to dividing the architectural work through iterations, which facilitated architectural decision-making. However, further research is needed to fully understand its impact across different contexts. On the other hand, the team identified the learning curve and process adjustments required for ArchHypo’s adoption as significant challenges that could hinder its widespread adoption. In conclusion, the evidence found in this study indicates that the technique has the potential to provide a suitable way to manage the uncertainties related to software architecture, facilitating the strategic postponement of decisions while addressing their potential impact.

Index Terms—Hypotheses engineering, software architecture, handling uncertainty.

I. INTRODUCTION

CHANGES in architecture are still considered bad and dangerous by several professionals, and even adopting agile methods, this kind of change is not welcome. Uncertainties

are not usually explicitly recognized, creating false certainty regarding quality attribute requirements [1] and the solutions used. This work 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. As an analogy, this perspective shift is similar to the Agile Manifesto¹, which, instead of fighting against changes in software requirements, proposed using techniques to better deal with them.

Despite the lack of software architecture practices at the beginning of the Agile movement, more recently, the terms Agile Architecture [2] and Continuous Architecture [3] have become more frequent in the literature. Two of the principles for Continuous Architecture state to “delay design decisions” and “architect for change.” However, defining which decisions could be delayed or where the design can benefit from a more flexible solution is not trivial. In fact, time is an important dimension when designing an architecture [4], and planning to distribute its decisions and development through the iterations is crucial, especially for software solutions that are in constant evolution.

This paper focuses on a technique named *ArchHypo* to identify and manage uncertainties related to software architecture. It is based on hypotheses engineering, a parallel for requirements engineering, encompassing techniques to handle hypotheses to guide experiment-driven software development. ArchHypo evolved from the refinement of a set of patterns that document agile practices for software architecture [5], [6], which have been disseminated to the software development community in talks at several international events and used in several projects in the industry. The first written report on the use of this technique shows how it impacted architectural decisions in a software startup over a period of 10 months [7]. In this experience, the technique was used for decision-making, but it was not incorporated into the development process.

Our objective is to propose ArchHypo as a technique integrated into the development process to manage uncertainties related to the software architecture. To achieve this goal, we adopted the design science methodology [8]. In this context, this paper describes the target artifact, ArchHypo, in detail and presents an empirical evaluation of it through its application in a critical project in a Brazilian software development company. This study aimed to understand the impact of the ArchHypo

Received 23 February 2024; revised 6 December 2024; accepted 16 December 2024. Date of publication 19 December 2024; date of current version 13 February 2025. This work was supported in part by FAPESP under Grant 2023/14646-1. Recommended for acceptance by Y. Dittrich. (*Corresponding author: Jorge Melegati.*)

Kelson Silva and Mauricio Ferreira are with the Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos 12227-010, Brazil (e-mail: kelson.silva@inpe.br).

Jorge Melegati, Xiaofeng Wang, and Eduardo Guerra are with the Free University of Bozen-Bolzano, 39100 Bolzano, Italy (e-mail: jorge.melegati@unibz.it).

Fabio Silveira is with the Federal University of São Paulo (UNIFESP), São José dos Campos 12247-014, Brazil.

This article has supplementary downloadable material available at <https://doi.org/10.1109/TSE.2024.3520477>, provided by the authors.

Digital Object Identifier 10.1109/TSE.2024.3520477

¹<https://agilemanifesto.org/>

technique in a real and challenging project, identifying the benefits of its use and the obstacles to its adoption. We collected data from several sources, including hypothesis assessment and evolution, architectural tasks included in the application backlog, and an open questionnaire with all team members. Evaluating the results, we could observe how the hypotheses were handled throughout the project, leading to a distribution of architectural work throughout the iterations. Additionally, different sources of evidence indicate that it contributed to the project's success. On the negative side, changes in the way of working and the learning curve were the main challenges to overcome in the technique's adoption.

II. AGILE APPROACHES FOR SOFTWARE ARCHITECTURE

Agile methodologies propose several practices to improve code quality and reduce the costs of changes. In test-driven development (TDD), developers create unit tests before production code in short cycles, focusing on creating the simplest solution for the current test suite [9]. The act of decoupling the classes for unit testing increases modularity, and the automated tests created are further used for regression testing [10]. Refactoring is another essential technique in the agile context. By continuously restructuring the code to improve its quality without changing its behavior, agile development teams can maintain a good design favoring changes and the application evolution [11].

These techniques are essential to keep the code ready to change according to customer feedback and to deliver software in small cycles. However, this claim might not be valid when changes in the architecture are involved. Some architectural decisions, e.g., the distribution of physical components, layer division, and framework use, can be so entangled in the code that any unanticipated change can be impractical [12]. Consequently, several teams adopting agile methodologies practice an upfront design for the architecture [13], limiting the kind of change that can be embraced. Since the identification and documentation of requirements in an agile project occurs throughout the project, the information available at the beginning of the project is usually insufficient for all the necessary architectural decisions.

Although the beginning of the “agile movement” can be traced back to 2001 with the publication of the Agile Manifesto, a study carried out almost ten years later (2010) revealed that existing agile methodologies describe very sparsely how to handle architecture [14]. It also showed a lack of scientific support for agile claims and the need for more agile practices focused on software architecture, as shown by existing evidence based on practitioners' experience. Indeed, in the same year, an industry report [15] advised agile practitioners to find the right balance for architecture work, making architectural decisions at the most responsible moment and not at the last possible one. However, a systematic approach for effectively integrating architecture within agile practices had yet to be developed. In 2013, Zdun et al. [16] presented practices for sustainable architectural decisions based on a case study. These practices

focused more on documenting, justifying, and tracking decisions, not enabling an evolutionary architecture. In other words, they aimed to sustain the decisions, making their respective rationale reachable and reproducible for similar cases, but not necessarily the architecture sustainable.

A grounded theory study conducted in 2015 interviewed 44 participants, including experienced architects, senior developers, team leaders, and development managers [13]. As a result, most participants believed that an agile team must find an appropriate trade-off between a full up-front architecture design and an emergent design. However, no technique was suggested to guide developers in that direction. Furthermore, a systematic mapping study published in 2016 [17] investigated 54 publications related to agile and architecture. None of the techniques mentioned in the studies had been widely used. The most frequently used approach, “Iterative Architecture Approach”, was discussed in only four studies, and most of the approaches (74.4%) were used in only one study. These studies underscore the lack of consensus in both academia and industry on how to handle software architecture in agile projects.

Moreover, more recent publications mention that working with software architecture in an agile environment remains challenging. In 2018, Hasselbring [18] reported that the tension between the agile and architecture communities was still reasonably high. The authors observed runtime adaptable models and techniques to keep architecture knowledge up-to-date for long-living software systems as critical issues for properly integrating architecture work into agile software development.

Building upon this, a 2019 paper by Dragičević and Bošnjak [2] delves deeper into these complexities, identifying the balance between agility and architecture as a key challenge. The paper advocates for an evolutionary design approach incorporating continuous and lean principles prioritizing quality attributes.

Yong [19] discusses agile software architecture and the necessity for a lightweight, flexible strategy to meet shifting requirements. Pang [20], [21] applies software engineering design concepts to agile architecture, emphasizing the requirement for a robust architecture that allows iterative and incremental development in large business IT systems. The articles emphasize the necessity for flexible and adaptable software architecture in agile contexts and the changing nature of software development.

The terms “continuous architecture” [22], “continuous architecting” [23], [24], “evolutionary architecture”, and “lean architecture” [2] are used in the context of continuous software development and refer to a constant activity of architectural design. Evaluating works that used that term, the conclusion is that these terms can be used as a synonym for the term “agile architecture” [25], as they refer to the same kind of practice. An interview study conducted with five large software companies [26] confirmed that the current practices of agile development methods are not enough to support the continuous evolution of the architecture. In conclusion, agile methods did not originally cover architectural practices. Furthermore, no consolidated approach has been accepted by industry or academia until now. Designing sustainable architecture poses a

significant challenge, especially in domains where innovation is a part of business.

III. HYPOTHESES ENGINEERING

In the last decade, there has been a growing interest in experimentation in software development. This concept 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 [27]. It comprises several techniques, such as iterations with prototypes, gradual rollouts, controlled experiments, or even problem and solution interviews. Therefore, in this context, the word “experiment” is not bound to scientific meaning, but points to a broader concept of data-driven rather than opinion-based decision-making.

Since this phenomenon has emerged in the industry, researchers have proposed models to describe how companies engage this concept in their development process. Two examples are the HYPEX (Hypothesis Experiment Data-Driven Development) model [28] and the RIGHT (Rapid Iterative value creation Gained through High-frequency Testing) model [29]. When comparing these models, a study [30] showed that they share a typical order of steps. First, based on the customer, user, and market’s current understanding, the team identifies, specifies, and prioritizes hypotheses to be tested in the process. Then, the team designs an experiment to evaluate one or more hypotheses, followed by the execution of the experiment and data collection. Finally, it analyzes the results, updates the hypotheses, and returns to the first step based on acquired learning. The first step consisting of identifying, specifying, and prioritizing hypotheses was called Hypotheses Engineering (HE), which is parallel to Requirements Engineering, given its role in guiding software development employing experimentation. The proposal of HE emphasizes the importance of these hypotheses in guiding the entire experimentation process. It is important to stress that, as mentioned above, experiments are not restricted to controlled experiments and, therefore, hypotheses are considered as a broad term related to assumptions and not only as suppositions or proposed explanations to be validated using statistics.

A grey literature review [31] identified techniques for handling hypotheses in the context of software startups, which were possible to classify into five activities: elicitation, specification, prioritization, analysis, and management. A recent study developed by our research team reports a technique named *HyMap* to identify initial hypotheses of a software startup using cognitive mapping [32]. Based on the resulting map, it is possible to systematically create hypotheses regarding these dynamics. Although good results were obtained by identifying hypotheses related to functional aspects, this technique does not focus on system architecture aspects.

IV. RESEARCH METHOD

Centered on discovery-through-design, the main focus of Design Science Research (DSR) is on problem-solving by designing innovative artifacts to meet practical needs identified

from a given domain [33]. The term “artifact” used herein is broadly defined as constructs (vocabulary and symbols), models (abstractions and representations), methods (algorithms and practices), and instantiations (implemented and prototype systems). DSR has its roots in the sciences and engineering of the artificial [34]. In the past years, DSR has been widely adopted by the Information System community and associated areas, such as Software Engineering and Computer Science. DSR is commonly used in these areas to produce new ideas to improve people’s and organization’s ability to adapt and succeed in constantly evolving environments in the scientific and technological paradigms [35]. This work applies the DSR framework as presented by [36] to propose the ArchHypo technique and evaluate its usage.

A. Design Science Research Rationale

Our research design is based on the seven guidelines for DSR proposed by Hevner et al. [36] presented below.

G1. Design as an artifact. DSR must produce a viable artifact in the form of a construct, model, method, or instantiation. It is satisfied because the target artifact of our study is a software development technique.

G2. Problem Relevance. The objective of DSR should be to develop technology-based solutions to important and relevant business problems. This requirement is fulfilled by the importance of managing uncertainties in software architecture design.

G3. Design Evaluation. The utility, quality, and efficacy of a design artifact must be rigorously demonstrated with well-executed evaluation methods. To this aim, we evaluated the artifact was performed in a real project, as described in Section VI, collecting evidence regarding the technique use from several sources.

G4. Research Contribution. An effective DSR must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies. This requirement was fulfilled by the fact that approaches based on hypothesis management are innovative and their investigation generates research contributions in the field of software engineering.

G5. Research Rigor. DSR relies on rigorous methods in both the construction and evaluation of design artifacts. To this aim, we employed research rigor in the design process and evaluation as detailed in Sections IV-B and VI.

G6. Design as a search process. The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment. In this regard, the design process followed for the technique construction is described in Section IV-B. We also stress that it is based on existing hypothesis engineering approaches and patterns for software architecture practices.

G7. Communication of research. This guideline states that DSR must be presented effectively both to technology-orientated as well as management-orientated audiences. To this aim, besides the current research paper describing the technique and reporting the evaluation results, we have performed

presentations describing the technique at conferences for audiences from both industry and academia.

B. Target Artifact Design

The target artifact is ArchHypo, a method based on hypotheses engineering to manage uncertainties related to the software architecture. It emerged from a set of patterns related to software architecture practices used in agile projects [5], [6], further incorporating concepts related to hypothesis engineering [27]. Furthermore, the patterns and ArchHypo were shared with the community through presentations at events and professional training and used by projects in the industry (more details in Appendix I). ArchHypo is described in Section V.

The first documented usage of this technique was in a software startup [7], in which the architectural hypotheses helped guide the evolution of the software architecture. In this experience report, after 10 months from the identification of the hypotheses and the elaboration of the technical plan, an evaluation was conducted of how they influenced the architecture. In this case, the architectural hypotheses were used as a guide by the startup founder but were not directly used in the development process. This experience happens in parallel with the one reported in this paper, and the results of one did not influence the other. Therefore, the study presented in this paper is part of broader work aiming at the development of ArchHypo.

The introduction of ArchHypo in the company development process started with an analysis of the process in place, followed by an initial proposal of adaptations to include ArchHypo activities. A new version of the company process was presented to the project participants in a training workshop, and some improvements were made based on the feedback received. This final process was then reviewed, documented, and disseminated internally in the company. This process is described in more detail in Section VI, together with the evaluation method.

V. ARCHITECTURAL HYPOTHESES FOR UNCERTAINTY MANAGEMENT - ARCHHYPO

This section presents ArchHypo, a method based on hypotheses engineering to manage uncertainties related to the software architecture. This approach aims to provide a more systematic way of managing the uncertainty related to the software architecture during a software project, enabling the distribution of decisions and architectural tasks through its iterations. This approach contributes to implementing two principles for Continuous Architecture [3], [22]: i) “delay design decisions” and ii) “architect for change”. ArchHypo does not aim to control all architectural activities and decisions since other existing techniques can be applied when there is no uncertainty involved.

A. ArchHypo Overview

The first step is to identify hypotheses that represent uncertainties related to the software architecture. This uncertainty represents a lack of information, which would be necessary for an architectural decision. They can be related to requirements

but also to current or candidate solutions. The identified hypotheses are then evaluated according to their uncertainty level and impact. While the uncertainty level reflects how far the team is to refute or accept the hypothesis, the impact represents the effort needed to change to a different alternative. This assessment is relative to the other hypotheses and is performed qualitatively by the team, normally using a three- or five-point scale.

Based on the assessment, the team should create a plan to handle each hypothesis, usually with the aim of reducing the impact or uncertainty. For example, isolating the part of the system affected by the uncertainty is a way to reduce the impact, and acquiring more information that can aid in decision-making is a way to reduce uncertainty. The steps of each technical plan are considered in each iteration planning, and the result of its execution is used to update the hypothesis evaluation. When there is no more uncertainty, the hypothesis is removed. The team should also discuss whether new actions are needed in the technical plan.

Postponing an architectural decision because of uncertain factors can by itself be a decision that can bring negative consequences. While ArchHypo does not aim to be a technique that will prevent them completely, it provides mechanisms that expose explicitly the uncertainties and support the team in evaluating the respective trade-offs in identifying the most responsible moment [5] for that decision. For instance, in cases with a high impact, the team might choose to make the decision as soon as possible. Additionally, when the decision is delayed, ArchHypo also suggests techniques that empower the team to reduce the impact of possible negative consequences.

B. Architectural Hypothesis

An architectural hypothesis represents any uncertain statement relevant to the software architecture design. According to the definition of hypothesis, it is based on limited evidence, and the goal is to use it as guidance for further investigation. An important characteristic is that it should be falsifiable, i.e., the possibility of proving it false exists. Unlike a requirement statement, which is usually assumed as true when written [1], by using the word “*hypothesis*”, it becomes clear to the whole team that it represents uncertainty.

Since ArchHypo focuses on hypotheses that can influence architectural decisions, their source can be directly or indirectly related to uncertainties in quality attribute requirements or the solutions involved in this decision. The first case is when the requirement is based on limited evidence or lacks some important details. For instance, the requirements for data storage might not be known due to the lack of information on the system usage. Moreover, another example is that the system should import data from another source, but the details of the format and quality of this data are still unknown. This uncertainty in the requirements can be related to future business possibilities, considering that it is not desirable to make a decision that would limit future choices. The second source of uncertainty is related to unknown consequences of the current architecture components or to the candidate solutions being considered. An example of

a hypothesis might be whether the cloud provider where the application is deployed is compatible with the component being considered for cryptography.

The hypotheses can be formulated as simple and direct statements that express the current beliefs of the development team. Some works propose templates to formulate hypotheses [37] stressing, for example, the need of an interrogative voice for hypotheses. We do not see any incompatibility in adopting them in the context of this technique, however, in our experience with the technique, we preferred to write the hypothesis more freely to reduce the friction in the adoption of the technique.

In the context of this technique, we highlight that the concept of hypothesis differs from requirements. While a hypothesis can directly relate to a requirement, that would not always be the case. For instance, in a requirement that states that the system should be able to process 1000 simultaneous requests, the uncertainty lies in the requirement itself if this number is unknown. Conversely, in an authentication requirement that states that it should use an existing single sign-on component, the uncertainty could lie in its compatibility with the access control framework. In this second case, the hypothesis will be related to the requirement but different from it. In other cases, the hypothesis might be related to comparing solution alternatives or refactoring possibilities, which may not even be connected to a particular requirement.

Uncertainties are frequently associated with risk, used in techniques such as the risk-based approach for quality requirements [38]. However, there is an essential difference between them. Although risk represents the probability and impact of a given event, uncertainty represents the lack of information to make a decision. For example, in the case of a hypothesis about the direction of the business strategy or the suitability of a set of solution candidates, there is not necessarily an unexpected event involved but rather a need for more information for decision-making.

Finally, to clarify another possible misconception, a hypothesis does not directly represent an architectural decision. Even though hypotheses are important for architectural decisions, it is imperative to note that these concepts are distinct. For instance, in a requirement about an external system that should be accessed through a new protocol, the hypothesis might be related to its compatibility with the transaction management component being considered. In this case, the hypothesis differs from the decision to use the component, but it indeed should be considered when making this decision. Furthermore, the techniques used to handle this hypothesis might lead to decisions about how this component should be integrated into the architecture, not as a definitive decision about its adoption but as a way to reduce the impact of a possible future replacement.

Given the difference between the architectural hypothesis and the other concepts highlighted in the previous paragraphs, ArchHypo differs from other existing approaches. This technique considers that uncertainty can be accepted and managed, guiding the focus of architecture tasks along the project. When uncertainties are explicitly stated and communicated, the development team deals with them in a different way, planning

actions to prevent them from bringing negative consequences to the project.

C. Assessment of Uncertainty Level and Impact

After elicitation, each hypothesis is assessed regarding its uncertainty level and impact. These two assessments are independent of each other and aim to provoke a reflection on the hypothesis, allowing a qualitative comparison with the other hypotheses and helping to choose techniques for handling it. This assessment does not aim to evaluate these characteristics precisely, so a simple three-point scale (Low, Medium, High) or a five-point scale (adding Very Low and Very High) can be adopted. Other hypotheses can be used as a basis for a relative score.

As mentioned before, the uncertainty level reflects how far the team is from refuting or accepting the hypothesis. For example, in a hypothesis regarding the feasibility of integration with a third-party system, the uncertainty would be high if the team had no information about the protocol to access the data. Accordingly, it would be lower if it is known that it uses a REST-based API, but there is little information about its authentication mechanism. It is essential to note that the uncertainty level differs from probability, which is commonly used for risk assessment. Both a very high and a very low probability would reflect that the team is close to more accurate information that can be used for decision-making, which means low uncertainty. Consequently, the uncertainty would be high if there is a lack of information to estimate this probability or when the alternatives had a similar chance to happen.

On the other hand, the impact represents the consequences and costs when there is a need for changing to a different alternative. The development effort is a common factor to be considered when evaluating the impact, but others, such as costs with infrastructure and loss of income, might also be considered. In a hypothesis about the number of simultaneous users that the system needs to support, the waste of resources for the infrastructure might be considered for the impact in the case of a lower number of users. Moreover, system slowdown (with its negative consequences for the business) and the effort to migrate to a more suitable infrastructure are types of impacts to be considered in case of a higher number.

D. Elaborating a Technical Plan

An important part of ArchHypo is to create a plan for handling the hypotheses. This plan comes from the pattern *Plan for Responsible Moments* [5] and following the same terminology, we use the name *Technical Plan* when referring to it. The plan can include one or more actions that might aim to: (a) definitely accept or refute the hypothesis; (b) reduce the uncertainty; (c) reduce the impact; or (d) define a criteria to postpone its handling.

The hypothesis can be accepted or refuted when the uncertainty is completely removed. When that point is reached, the architectural decision can be made following the existing techniques. An approach based on “trial and error” is implementing a candidate solution. If this solution is tested and the respective

uncertainty is removed, the team does not need to handle that hypothesis further.

Another approach is to reduce the uncertainty by obtaining more information regarding the hypothesis in a way that allows the development team to proceed with more confidence. Diverse types of experiments [39], such as customer surveys, A/B experiments, and fake door tests, can be employed to validate requirements, identify consequences, and compare solution alternatives. Software analytics approaches that bring data-driven decision-making into software engineering processes [40], [41] are helpful when the software already has a version where measurements can be collected. It can assist in identifying relevant metrics about a given quality attribute of the system that can provide insights about the target hypothesis.

Architectural Spikes and Tracer Bullets are patterns documented in the pattern collection that originated this technique [6]. They propose different ways of exploring solutions, also providing a way to reduce uncertainty. Architectural Spikes are small technical experiments created to prove or disprove feasibility or to evaluate a property of a given solution. On the other hand, a Tracer Bullet is an architecturally significant feature that helps to exercise and demonstrate an end-to-end path inside the architecture, aiming to evaluate how new technologies could be integrated. While in the spike, the problem is explored in an isolated environment, when a Tracer Bullet is developed, the team members use a feature inside the target system as a reference for experimentation.

Besides the alternatives to interfere with the uncertainty, there are also techniques that can be used to reduce the impact. For example, by adopting a modular structure, it is possible to isolate in a component the part of the code that would be affected by a hypothesis. Several patterns can be used in that direction. For instance, Anti-corruption Layer [42] can be used to reduce the impact of uncertainties regarding the integration with a legacy code. Moreover, aspects, dynamic proxies, and interceptors [43] can separate a crosscutting concern related to a hypothesis from the rest of the code. Isolating what would potentially be affected by the rest of the application facilitates changes in the future, either by refactoring or replacing that part.

Other flexible solutions can also be adopted to reduce the impact. For instance, deploying the application in a cloud environment that provides elasticity is an alternative to deal with uncertainty in the application load [44]. Using self-adaptive architectural solutions [45] is another alternative in that direction. Since flexible solutions usually have a higher associated cost, the hypotheses can help to evaluate the trade-offs in adopting this kind of solution early in the project.

Another type of action that can be assumed is to make small modifications in the software engineering activities to implement guidelines that can reduce the impact or the uncertainty level. Examples of types of guidelines are: (a) include people with certain knowledge in activities or decisions; (b) retrieving some additional information before an activity; (c) executing some test or validation after an activity; and (d) adding constraints or recommendations on how an activity is executed. For example, before the inclusion of new libraries in the project, a

guideline could include an activity to retrieve information about its compatibility with cloud providers. As another example of a guideline, it might define that a security specialist should be included in the design of solutions that include data exchange among different systems. The implementation of the guideline tailors the development process with small adjustments focused on handling the hypotheses, allowing the team to gather more information to reduce the uncertainty or take precautions that can decrease the impact.

The last approach that can be used to handle the hypothesis is to use strategies that define criteria to decide when they should be handled. For instance, Architectural Trigger [6] defines a condition based on a quality attribute that defines when the team should pay attention to a given hypothesis. For example, a trigger related to the application's scalability can use the number of simultaneous accesses. Another pattern that can be adopted is Agile Landing Zones [46], which are a set of criteria used to monitor and characterize the "releasability" of a product. They are similar to release criteria with tolerances in acceptable values, giving some flexibility in defining what is "good enough". Having a Agile Landing Zone close to the minimum value can be used as criteria to move forward in other strategies for handling a related hypothesis.

This work is aligned with the idea that good solutions come with combining technical expertise with creativity [47], [48]. So, these techniques are not designed to provide ready-to-use recipes but simply to point in the right direction. For example, there is much room for creativity when finding a flexible approach to reduce impact or suggesting process small adjustments that can reduce uncertainty.

VI. EVALUATION METHOD

This study aims to evaluate the usage of ArchHypo in a realistic context, so we are guided by the following research question:

How does using ArchHypo for architectural uncertainty management impact a software development project?

To answer this research question, we introduced ArchHypo for a development team to use it in a real project and evaluated its usage's consequences. For this study, we did not consider any initial assumption and followed an inductive approach to identify the main benefits and challenges of the technique usage [49], [50], [51]. The suitability of this evaluation approach is justified by the real-life context being considered. Additionally, the boundaries between the phenomenon and the context are not clearly evident [49], which in this study are respectively ArchHypo and the software development context, such as team, company, project, and processes.

The study started with an exploratory phase, in which we analyzed the company's current process entry points, proposed process changes to use ArchHypo, and reached an agreement on the changes with the development team. Additionally, we held a workshop to train the team on using the technique. In the next stage, we selected a project to implement the adjusted process.

We collected data about the software development activities planned and executed during the project and just after the product's first release. The collected data also included the hypotheses, their assessment, and their respective technical plan activities. Additionally, we designed and distributed a questionnaire to the project team to identify and extract more detailed data and insights into the technique's effects according to the team members' perceptions.

A. Company and Project Description

The company where the evaluation study took place is named Jetsoft, described in more detail in Appendix II. The reasons for this choice were: (a) it adopts an agile process compatible with ArchHypo; (b) one of the authors had the autonomy to make changes in the development processes; (c) there was a mission-critical project in which the technique could be applied.

In May 2022, due to the request for changes in the structure of budget accounts, the development of a new version of the Budget Proposal system of the Planning and Management Secretariat (PropSEP) was required. Its mission is to receive and integrate all data from other systems participating in the budget and provide functionalities that allow for the budget's analysis, definition, formalization, and approval. Initially, requirements were established for structural modifications to the data model, technological evolution to improve usability and user experience, and improved performance during data integration and loading. Finally, the deadline for the new version to go into production was the beginning of August 2022, guaranteeing all the reliability and interoperability required by the production environment. As a reference, the approved budget of the State of São Paulo, processed by this system for the 2023 fiscal year, was 317 billion Reais (BRL)².

The context in which the system is inserted presents several challenges related to, for example, achieving strict deadlines imposed by the legislation, including decrees and amendments, processing and storing large volumes of data, and fulfilling rigorous quality requirements, such as usability, performance, security, and reliability. When we analyzed all the challenges imposed by the environment and the demands of the project requirements, we considered it an interesting scenario to implement ArchHypo.

All members of the development team graduated from computer science-related fields, and most of them had certifications in the technologies used in the project, such as Oracle, WebSphere, and GeneXus. Their individual experience is presented in Table III. The domain was not familiar to the team members since another team had maintained the previous application.

The development used a low-code platform called GeneXus, which generated the final application in Java. To have a dimension of the application size and complexity, at the end of the project, it was composed of 819 objects inside the platform: 288 procedures, 149 structured data types, 74 external objects, 73 transactions, 73 data views, 70 web panels, 65 data providers, and 22 subtype groups. All of these objects have associated

code created in a proprietary language of the platform. Only the dedicated application database was composed of 73 tables that had 1390 attributes and 105 indexes. It is important to note that even when developed on a low-code platform, the application has complex rules and a significant amount of code. The architecture uses a MVC-based layered architecture that explicitly separates concerns about visualization, business rules, and database access. The view is implemented by web panels, the controller by procedures and data providers, and the model by transactions and structured data types. Modularization was also implemented in a functional dimension, separating the elements from small groups of features.

A relevant contextual factor that brought complexity to the project was related to the requirements for interoperability with other systems, which were maintained by other teams and used different technologies. The system interoperates with seven other systems using the following strategies: access to REST services, direct access to the application database, and the generation of scripts for the initial data load.

B. Integration of ArchHypo Into the Process

The study began on May 1st, 2022, with an assessment of the process in place at the company before our intervention and by informally collecting feedback from the team. To this aim, we held weekly meetings with the team for a month. This step was essential to understand the scenario better. During data collection, one of the observations was that the process already encompassed some practices and activities to deal with architectural requirements. We then created an analytical report with the main challenges and adaptations to include ArchHypo activities. The analysis served as input to create a new version of the process with the inclusion of the necessary modifications.

Subsequently, we held a one-day ArchHypo Training Workshop with all project participants to present the updated process. We carried out training sessions and incorporated some points of view and improvements as they emerged. Finally, the process model was validated and internally disseminated as a guideline to support development.

Following this, we met with those involved to understand better the project's characteristics. We evaluated the challenges posed by quality attribute requirements, such as performance, security, reliability, and usability, in addition to the constraints of a two-month development schedule.

On May 30th, 2022, one of the researchers participated in the Requirements refinement meeting ceremony, which lasted approximately two hours, facilitated by the Scrum Master, in the presence of the Product Owner (PO), the Architect, and the Developers. The Backlog, which consisted of functional and quality requirements, was refined and prioritized, and its items were estimated. The Architect employed a shared spreadsheet to record the hypotheses as they were defined. The following day, one of the researchers engaged in a Sprint-0 Kick-Off Meeting, a ceremony that lasted approximately eight hours. During this meeting, the PO detailed a priority backlog item, and the architect presented the identified architectural hypotheses and assessed their uncertainty level and impact. Subsequently, the

²equivalent to 60 billion USD as of December 29, 2022

team established the technical plan for each hypothesis. The hypothesis information was kept in a spreadsheet in a shared folder available to the entire team. In parallel, the developers added the respective tasks to the Kanban board implemented on Trello to refine them later. These tasks received labels regarding the respective hypothesis and the technique employed (among those described in Section V-D).

On June 1st, 2022, the team started the project's sprints, which lasted seven business days each. At the beginning of each sprint, the team participated in a Sprint Planning, in which the architectural tasks were prioritized in the backlog. Some tasks that were out of the development team's scope, like the ones that required gathering more information with the clients, were assigned to the business analyst. During the Daily Meetings, the developers updated the status of these tasks and discussed possible adaptations when needed.

At the Sprint Review ceremony, the team discussed the quality of the package delivered, followed by the Sprint Retrospective, where they discussed how the process was carried out, including the adaptations to ArchHypo. Finally, at the Quality Management Meeting Ceremony, where the PO, the Architect, and the Scrum Master were present, the entire team discussed the result of the executed tasks and reevaluated the uncertainty level and impact score for each hypothesis accordingly. The team removed hypotheses when their uncertainty reached a minimal level. For the remaining hypotheses, the team discussed adjustments in the technical plans.

The project consisted of six sprints until its completion on July 28th, 2022. On August 1st, 2022, the final version of the product was released into the production environment as planned. Throughout August, one of the researchers participated in the Product Support Service, and the system was monitored, seeking to identify non-conformities and improvements in incidents reported by users.

C. Data Collection

As part of the evaluation method, we used of multiple data collection methods, as suggested by Wohlin and Rainer [51]. First of all, we collected all the work by-products created or other artifacts used during the project execution, including documents directly related to ArchHypo. The artifacts collected were:

- **project management documents:** digital Kanban board (Trello) history, user stories, incident reports, burndown charts, and product owner task log;
- **communication artifacts:** e-mails and instant messaging (Skype) conversation history;
- **ArchHypo artifact:** spreadsheet containing the hypotheses and their evaluations throughout the project.

Besides that, the first author of this paper participated in the project ceremonies, i.e., he was a participant-observer. In this data collection method, "the researcher essentially becomes part of the team and participates in key activities" [52]. The choice of having the participant-observer was to ensure that the ArchHypo method was adopted and used correctly. We also understand that this choice has a trade-off: a direct interaction

between the researchers and the team can introduce a researcher bias. To avoid that, we kept the interference to the minimum and focused on clarifying doubts and guiding the team on the correct usage of the technique. Moreover, the results also discuss the learning curve and the autonomy of team members in the application of the technique.

Finally, at the end of the project, an open-ended questionnaire was administered to understand the participants' perceptions of the technique. The questions asked the participants: (a) to describe the technique, (b) how it differs from the way they used to work, (c) what benefits and difficulties it brought to the project, team, and product, (d) how it affected other practices, and (e) if they intended to use it in another project and why. The answers aim to provide a view of the changes brought by ArchHypo compared to the participants' previous experiences.

The company gave the researchers full access to all the above-mentioned documents and authorized the team members to freely discuss their experiences using ArchHypo. Although the company allowed this material to be used in the study and to present the hypotheses and the respective technical plan elements, the researchers were not authorized to make the complete participants' answers publicly available.

D. Data Analysis

Data analysis combined a systematic analysis and sense-making of the collected documents and a thematic analysis [53] for the answers to the questionnaire.

For document analysis, the first step was to become familiar with the data. To achieve this goal, the first author, who was also the participant-observer in the project, explored and noted down ideas, read and re-read the six versions of the hypothesis recording spreadsheet, making it possible to revisit the classification and evolution of statuses, level of uncertainty and impact. He also considered the notes regarding the technical plans, seeking to better understand the validated scenario, the descriptions of the hypothesis, and the sprint planning. In the next step, he browsed the product backlog notes the PO recorded, organizing the architectural items. He thoroughly examined the history of tasks imported from the Trello tool used by the team, classified the support history by the type of incident reported by users, and identified which quality attribute was referenced. Finally, he scanned the PO action history and categorized the refinement tasks with the customer regarding the hypothesis and the quality attribute met. Based on that, the data was grouped by characteristics and relevance, with 64 items identified: 10 hypotheses, 6 quality attributes, 7 architectural practices, 6 sprints, 35 tasks, 6 PO actions, and 46 support reports.

To analyze the support reports, we individually read the problems reported by the users and searched for issues that could be related to any of the hypotheses. In cases where a connection was found, we went deeper into the problem and solution for that support item to check if it was related to the uncertainty represented by the hypothesis. Since the company does not adopt a classification for the support reports, we considered the dates on which they were created and solved as a reference for their complexity.

TABLE I
OBSERVED EVENT TYPES DURING THE PROJECT
DEVELOPMENT AND MAINTENANCE PHASES

Events types	Quantity
Development phase	
Client validation meetings	10
Daily meeting	12
Informal collection of team feedback	7
Meetings to adapt the process to ArchHypo	4
Other on-demand meetings	13
Quality management meeting	6
Requeriments refinement meeting	15
Sprint planning	6
Sprint retrospective	6
Sprint review	6
Sprint-0 Kick-Off Meeting	1
Maintenance phase	
Other on-demand meetings	8
Product support service	30

For analyzing the answers to the questionnaires, we applied thematic analysis, which consists of a flexible process of six iterative phases to identify, analyze, and interpret convergences in qualitative data [53]. The six phases are: familiarizing yourself with your data; generating initial codes; searching for themes; reviewing themes; defining and naming themes; and producing the report [53].

VII. RESULTS

This section presents the results of the empirical study. It aims to present a general view of the data collected. It includes the list of observed events, hypotheses evolution, technical plan tasks, issues raised after the delivery date, and the codes extracted from the team members' responses to the questionnaire.

A. Participation in Events

Table I presents, in alphabetic order, a summary of the events in which the participant-observer [52] participated during the project. During these events, the author supported the team in the adoption and implementation of the technique and observed team dynamics, the environment, and the project challenges. From these observations, no data were collected. However, they were used to gain contextual knowledge to interpret the data obtained by other means.

The events are divided into two periods: the development phase, between May 1st and July 28th, 2022, and the maintenance phase, between August 1st and September 8th, 2022. Two types of events important for the adoption of the technique were the ArchHypo training workshop and the meeting to adapt the process to the technique, which occurred at the beginning of the project. The hypotheses were initially identified at the Sprint-0 Kick-Off Meeting and were further discussed and updated at the Quality Management Meeting, which took place at the end of each iteration. It is important to highlight that among the meetings classified as "on-demand", some were requested by

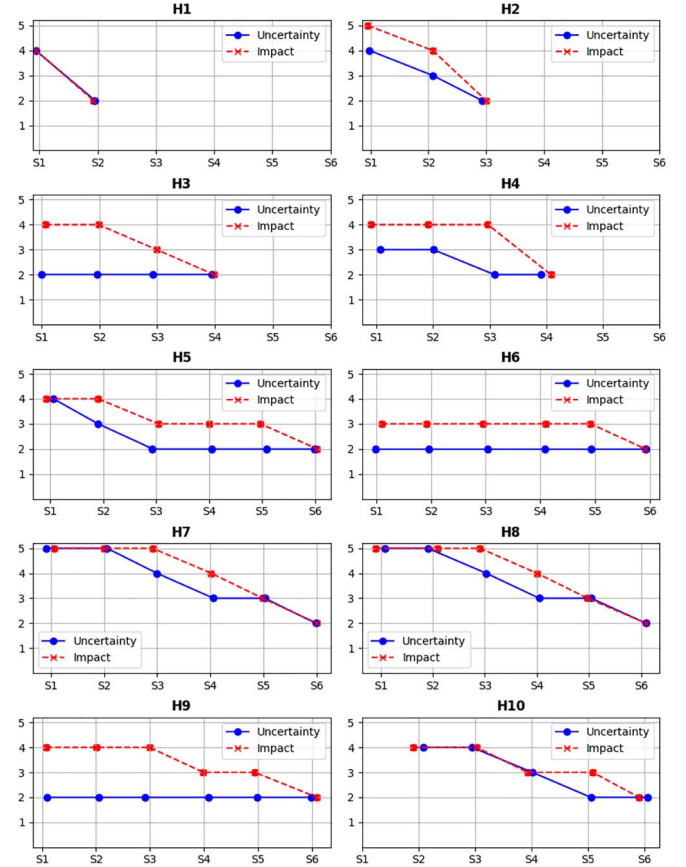


Fig. 1. Evolution of hypotheses uncertainty and impact levels across iterations.

the development team to clarify the usage of the technique. Another important point to clarify is that all iteration meetings, such as Spring planning and reviews, were covered. However, in more frequent ones, such as daily meetings, only some were observed.

B. Hypotheses Documentation

Throughout the project, we examined the management and evolution of hypotheses through spreadsheet documentation. Fig. 1 depict the progression of the hypotheses' uncertainty levels and impact assessments, respectively. The hypotheses are listed in Table II, and their respective details, like motivation and core technical plan elements, are in Appendix III. In the project, there are hypotheses related to six different quality attributes³.

We could observe that 50% of the hypotheses with the highest levels of uncertainty and impact were handled and solved in the first half of the project, i.e., at the end of the third sprint. This points to a focus of the team's efforts on items that are more important to the project's progress. We also found that two hypotheses, which could potentially have a high impact

³Team productivity refers to the speed at which a new feature can be developed in the system, which depends on several factors in the development process and software architecture.

TABLE II
HYPOTHESIS IDENTIFIED IN THE PROJECT

ID	Statement	Quality Attribute
H1	It is possible to process batches of 1Gb per minute using just one server.	Performance
H2	The components used in the system can be integrated with the government Single Sign-on protocol.	Security
H3	It is possible to identify any inoperability in the subsystems and deal with it without user intervention.	Reliability
H4	The integrity of data obtained from external systems can be assessed to avoid inconsistencies.	Reliability
H5	It is possible to maintain compatibility with both the old data format and the new one.	Flexibility
H6	With a short online training, a user of the old system will be capable of using the new one.	Usability
H7	The core functionalities of the system will be in production on the target date.	Team Productivity
H8	The core features of the system will not be unavailable after the system enters production because of critical bugs.	Reliability
H9	The client approval needed before implementing a solution will not become a productivity bottleneck.	Team Productivity
H10	It is possible to view and update data both from old and new format in the same user interface.	Flexibility

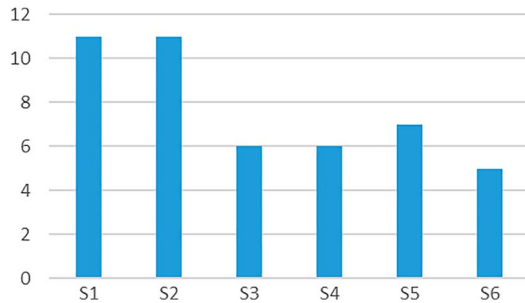


Fig. 2. Distribution of hypotheses technical plan activities by sprint.

on several system components, had actions from their technical plans being executed from the first to the last sprint.

C. Technical Plan Activities

After analyzing the team iterations backlog and the PO task list, we identified 41 activities related to the technical plan for the hypotheses. It is worth highlighting that six of these activities were executed by the PO, which played an important role when it required an interaction with the client. Fig. 2 presents the number of activities from a hypothesis technical plan performed in each sprint.

Each activity was labeled according to the type of technique used. Fig. 3 depicts the number of tasks related to each technique. As presented, software analytics and guidelines implementation were used more often. Considering the range of quality attributes related to the hypothesis and the variety of

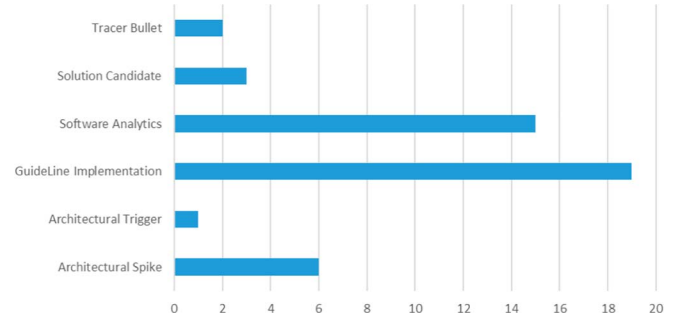


Fig. 3. Distribution of architectural practices in technical plan tasks.

techniques implemented, this project was considered suitable for evaluating ArchHypo.

D. Issues After Project Release

Users reported 46 issues during the maintenance phase. Of these, 21 were considered improvement requests, while 25 were identified as bugs. Considering the criteria to evaluate the issue's complexity, all of them could be resolved on the same day they were reported, indicating that the problems were straightforward to solve.

Regarding the quality attribute related to each issue, the one with the most occurrences is usability, with 28. An analysis of the issues revealed that they mostly focused on suggestions for improvements in the user interface or reporting a component presenting an unexpected behavior. Considering that the hypothesis related to usability (H6) focuses on the adaptation of the users from the old system, none of the issues was related to that.

The second quality attribute with the most occurrences was reliability, with 12 entries. Most of them described problems with functional features that did not prevent the usage of the system. One was related to data migration, which was the target of one of the hypotheses (H4). This problem was in the data migration script from the external system, which did not include all the data. Since the solution adopted to monitor the problems in the data obtained from external systems compared the received data with the processed data, it did not detect this problem that was in the script that sent the data. Since it was a simple issue, the team easily fixed it, but it was considered out of the system's scope.

The final quality attribute with entries was Security, with six of them. Four of them were issues related to access control, and the other two reported situations in which the system asked for authentication when a single sign-on token should be recognized. This issue was solved with a configuration in the production environment and was unrelated to hypothesis H2, which targeted the compatibility between the platform's built-in component and the single sign-on protocol.

This result was considered a success both by the team and the client since the system entered the production phase on the target date. In addition to the reported issues, none of them were considered critical and prevented the usage of the system. Furthermore, based on our analysis, none of the issues were

TABLE III
TEAM MEMBERS AND THEIR EXPERIENCE

ID	Role	Experience with Agile Software Development
P1	Technical Leader	13 years
P2	Developer III	8 years
P3	Product Owner	8 years
P4	Developer II	4 years
P5	Developer II	5 years
P6	Developer III	7 years
P7	Developer I	2 years
P8	Product Manager	13 years

considered to be related to any of the uncertainties represented by the hypotheses.

E. Development Team Experience

The project team comprised the following roles: product manager, product owner, scrum master, solutions architect, technical leader, and developers, as shown in Table III. In total, 12 professionals with an average of 13 years of development experience in agile software participated in the project. Of these, eight worked directly on executing the sprints and responded to our questionnaire.

Based on the questionnaires answered by eight team members, a total of 21 codes were extracted. We grouped them into three categories with respect to the aspects being considered: people, process, and product, as presented in Table IV. We also classified the codes into benefits or challenges. From these, 16 represent the benefits achieved by the technique. The other five codes are related to the limitations or challenges in their usage, with a greater weight attributed to the pillars of people and processes. Since all team members who participated in the project answered the questionnaire, we believe that these answers represent the team's view on the experience of using ArchHypo.

In Table IV, after the code labels are presented, two numbers are separated by a comma: the first represents the number of quotes extracted, and the second is the number of participants in which this code was present. An excerpt to exemplify each code is also presented.

F. Findings

Based on the level of hypotheses uncertainty assessed in each iteration by the team members, presented in Fig. 1, it is possible to see that for some hypotheses, the uncertainty was eliminated. In this case, these hypotheses were removed from the management as they do not reflect uncertainty. On the other hand, for some other hypotheses (H6, H7, H8), even with the adoption of some techniques, the uncertainty remained until the last iteration. This fact leads to the first finding.

Finding 1: *While some hypotheses are solved in the course of the project, for others, some uncertainty remains until the end of the project.*

When evaluating the hypotheses that were solved during the iterations, we observed that most of them capture uncertainties related to the lack of information from the team related to components or requirements. In this case, once the activities to gather this information were performed, the issue was clarified, and the uncertainty was eliminated. As an example, since H2 was related to the compatibility of a component with the single sign-on mechanism, once it was confirmed, the hypothesis was accepted. The hypotheses that remained until the end of the project were those that represented uncertainties about the impact of activities being performed by the development team on the quality attributes. For instance, the uncertainty related to team productivity expressed in H7 and about the adaptation of users from the old system in H6 were managed until the end of the project.

So, even with some techniques being used to manage the uncertainty level and impact and having them evaluated as low or medium, the uncertainty was not completely eliminated. When a new feature is introduced, these quality attributes can be affected, so it makes sense to keep these hypotheses monitored until the end of the project. This finding is relevant since it is evidence that some uncertainties remain until the end of the project and that, by managing the hypothesis, these uncertainties can be followed and managed.

Another consequence observed during the project was the distribution of the activities to deal with the hypothesis throughout the project iterations. According to what is presented in Fig. 2, it is possible to note that the first two iterations had a higher number of tasks regarding the software architecture, but that number remained stable in the following iterations. This piece of evidence points to our second finding.

Finding 2: *The hypothesis management allowed prioritizing and planning architectural tasks distributed in the project iterations.*

The tasks in the first iterations focused on activities that comprised the technical plan for hypotheses with high uncertainty or high impact. For example, if H1 was rejected, it could change how the software was distributed, generating a high impact. However, the hypothesis assessment also enabled the team to understand what could be postponed and planned for future iterations. For instance, when handling H4, even with a high impact, the team understood that it had a low interaction with the rest of the system and dealt with it only in the fifth iteration. According to four team members, the technique improved the process by allowing a better prioritization of the activities. One of them mentioned that it “*helped prioritize the backlog*” (P1). Another team member mentioned a practice to be kept in future projects: “*avoiding executing activities that depend on things with a high level of uncertainty*” (P7).

This better planning and division of the architectural tasks also positively impacted the project management, according to four respondents. Referring to the previous process used by the team, P3 mentioned that “*things happened catching everyone by surprise, and that impacted the project scope, delivery dates, and costs directly*”. Also contrasting with the previous

TABLE IV
BENEFITS AND CHALLENGES OF ARCHHYPO AS PERCEIVED BY THE PARTICIPANTS

Code	Excerpts
People	
Benefits	
Self organization (11,4)	“the technique showed whether the team should have a self-management attitude at the beginning of the project” (P2)
Technical capacity (6,4)	“benefited in the search for technical excellence in the development of the final product.” (P2)
Concentration on objectives (7,5)	“Using information based on discoveries and experimentation, we identify doubts and assumptions that lead to doing only what is necessary” (P4)
Expert decision making (16,7)	“It allowed more effective management that can make decisions to overcome risks” (P3)
Challenges	
Adapting to changes (15,8)	“initial adaptation/organization with the use of the new technique and difficulties in fitting in and prioritizing new non-functional tasks at the appropriate time” (P2)
Learning the technique (14,8)	“train the team a little more on how to find, map risk scenarios, and define the necessary actions for each scenario, allowing decisions to be made in time to overcome risks, would make the team less dependent on the team of architects. We had a workshop on your use of the technique, but I think it wasn’t enough” (P3)
Process	
Benefits	
Achievement of Goals (4,4)	“If there is uncertainty, experimentation helps measure and achieve key indicators” (P4)
Sustainable development (6,3)	“benefited from predictability, security, transparency, quality, and sustainable pace” (P3)
Efficiency of the development process (12,7)	“Hypothesis management had a positive impact on the entire project development cycle” (P1)
Project management (5,4)	“In this way, I consider it an important technique to evaluate, control, monitor and maximize project success.” (P1)
Risk management (23,8)	“the technique to formulate questions and transform them into assumptions along with experimentation helped reduce risks before the start and during the project progress” (P4)
Prioritization of work (5,4)	“benefited from clearer and more detailed visualization of delivery packages” (P2)
Reduction of production costs (3,3)	“allowed cost reduction and improvement in the final quality of products and processes” (P5)
Architectural tasks in development (5,4)	“collaborating on architectural decisions and practices in system implementation” (P1)
Expected behavior (3,3)	“brought as a benefit to the progress of the project the predictability” (P1)
Challenges	
Adjustments to process activities (2,2)	“difficulties in including new activities in the process that are not development” (P8)
Mapping risk scenarios (4,2)	“To have the answer to an assumption whether the path is good or not, we carry out tests and build features. During construction, other assumptions are raised that are sometimes not shared” (P4)
Product	
Benefits	
Customer satisfaction	“provided a clear vision of what should be prioritized, seeking to deliver the greatest possible value to the customer” (P6)
Meeting functional requirements (6,5)	“brought as a benefit to the final product a greater scope in quality and its added value” (P1)
Quality of the product delivered (8,5)	“the developed product met expectations regarding quality and value delivery proposed at the beginning of the project” (P6)
Challenges	
Increased development effort (5,4)	“in the backlog there were previously only tasks with main product functions” (P3)

process, another participant said that “*with the technique based on hypotheses, it became clear and transparent how the project was progressing*” (P2).

Related to this better planning, seven out of eight of them mentioned that one of the benefits of the technique is an improvement in the process’ efficiency. The documentary evidence showing that all hypotheses were successfully handled and how the tasks were divided in the iterations corroborates that claim. Furthermore, there was a consensus among the respondents that the technique improved the way project risks were handled. Six participants also mentioned the sustainability of the development as a positive consequence. Based on that, we extract the third finding.

Finding 3: *The team perceived an improvement in process efficiency and risk management by using hypothesis management.*

When pointing to an improvement in process efficiency, respondents mentioned that ArchHypo allowed the identification of points to be prioritized while having enough time to handle them appropriately. The word “*quality*” was mentioned by three participants, while “*safety*” was mentioned by two of them in this context. Another point that appears in the interviews due to this efficiency was “*delivery on time*”. Transparency was another important point since “*it was positive to*

provide more transparency during the whole process, allowing changes in the direction when necessary” (P3). Considering the context of previous projects where the team was required to work extended hours to adhere to deadlines, sustainable development was identified as an advantage by six participants. One individual noted that *“without [ArchHypo], timely delivery would have been unattainable without reducing the project scope” (P3).*

While the term “process efficiency” was mentioned by the participants, we cannot be sure if all of them considered the same meaning when mentioning it. Based on the quotes, it is possible to understand that safety, transparency, and predictability are some qualities considered when pointing to a process improvement. However, this result reflects the perception of the team since no measurement was performed to support that improvement.

All participants pointed out that the ArchHypo, even though focusing on uncertainties, helped to handle the project risks. Some participants highlighted how the technique allowed a vision of important things to be handled in the architecture: *“provide a more clear visualization of the risks that could threaten the final delivery of the project” (P2); “improvement of the visibility of the challenges that should be faced” (P7).* Moreover, other quotations also focus on the technical plans to handle the hypothesis: *“it makes possible to elaborate action plans for each of the scenarios” (P3); “the technique to formulate questions and transform them into assumptions, together with experimentation, helped reduce risks before and during the project” (P6).*

In this project, there were hypotheses related to six different quality attributes: performance, security, reliability, flexibility, usability, and team productivity. Different types of practices recommended to compose the technical plan were used to handle the hypotheses. This is represented in Fig. 3. Even if the strategies adopted followed the types suggested for the technical plan, the team found creative ways [47], [48] to use them to reduce impact and uncertainty. That leads to the fourth finding.

Finding 4: *Hypothesis related to different quality attributes were handled using different strategies in creative ways.*

As an example, the uncertainty represented in H1 reflected the concern that just one server would be enough for data processing. To handle it, a software analytics activity used the data from the previous year as a reference, together with the client projection on how much it would increase, as a strategy to reduce uncertainty. The technical plan also included a measure to reduce the impact, which was to structure the application so that the component that processes the data could be distributed later. Since uncertainty and impact were considered both high, the first action from the technical plan was planned and executed in the first iteration. Based on the measurements performed, the team concluded that the current infrastructure was enough to fulfill the performance requirements, and the action to reduce the impact was considered unnecessary.

Another interesting case was an uncertainty related to usability (H6). Users should use the system for mission-critical work right after its release, and given the user profile, the team was concerned about their adaptation to the change. The first action adopted by the team was to search for graphical components with a look and feel similar to the current system and try to follow a guideline to maintain the same user interface design style. In addition, early validation of the screens by real users was also introduced as part of the development process as a form of a lightweight experiment. Finally, architectural spikes were conducted to identify efficient ways to add help mechanisms and dynamic hints when using the system.

In the aforementioned example, it is possible to observe an instance of the technique referred to as “guideline implementation”. It was interesting to observe that it was the most used strategy to handle the hypothesis. By “guidelines”, we refer to small adjustments, like adding a new activity or providing recommendations in the execution of the current ones, that tailor the process to fit a specific need. This leads to the fifth finding.

Finding 5: *Defining a guideline that tailors the development process to a specific need was the most used strategy to deal with the hypothesis.*

H7 was an important hypothesis in this project focused on the uncertainty of whether the team would be able to deliver the software on time. As a complimentary issue, H8 focused on the uncertainty regarding achieving good software quality since delivering on time a product with several defects would not be useful. To balance these forces, the team adopted several guidelines. From a programming perspective, the guideline proposed a more efficient approach to creating functionalities inside the adopted platform. Considering the iteration planning, another guideline focused on how functionality should be broken and prioritized to avoid focusing on something that was not essential for the product release. Also, regarding team productivity, H9 expressed uncertainty regarding the interaction with the customer becoming a bottleneck. Accordingly, a guideline was agreed upon with the customer, defining which kind of decisions they should be present, giving the team more autonomy to decide. Finally, also focusing on quality, some guidelines targeted the automated testing approach, defining how the test should be created.

An important goal of the ArchHypo technique, which was confirmed by the development team, is the support for decision-making. Based on the answers to the questionnaire, the technique helped the team to identify important goals to focus on and provided guidance on their achievement. That confirmation is the sixth finding.

Finding 6: *The team recognized that the hypothesis management allowed better architecture decision-making.*

The answers of seven respondents mention an improvement in architectural decision-making. As a reason for this

improvement, it was mentioned that it “*makes the team base the decisions on information*” (P4), “*brought as a benefit the possibility for the team to experiment*” (P1), and that “*the technique helped make transparent risks and impacts*” (P5). Referring to the approach used in previous projects, it was said that team members “*were caught by surprise and could not select the best solution, but the first one*” (P1), that “*the focus was to solve the problem with the first solution found, not the most viable and intelligent*”, and “*that previously [decisions] were taken under pressure and driven by defects found*”. On the contrary, a participant mentioned that if one of the uncertain scenarios goes in an unexpected direction “*there are previously planned paths for the team to follow, avoiding surprises and fatigue*” (P3). One quote that well captured one of the technique goals was that “*it allowed a few decisions to be anticipated and delayed*”, reflecting that it supports the team in choosing a suitable moment for each architectural decision to be made.

Five team members mentioned focusing on important goals, which are important for decision-making, as a positive consequence of the technique’s adoption. In the participants’ own words, it “*directs team effort*” (P3) focusing on “*what really matters*” (P4) and “*tasks that are really needed to create the proposed solution*” (P6). Moreover, respondents not only highlighted the emphasis on important goals but also acknowledged the contribution of four individuals toward achieving these objectives. According to one of them, “*when facing uncertainties, the experimentation helps measure and reach key indicators*” (P4). It was also mentioned that the technique “*improved the final quality of products and processes*” (P5) and “*make possible to reduce costs and increase the delivery of value to the client*” (P6).

Corroborating this support in decision-making, we also found evidence of the technique’s contribution to product quality. From the issues reported in the month after the software release, we can find documentary evidence of the project’s success. Based on the reported experience from the team members, we have evidence that the use of the technique contributed to product quality and customer satisfaction. That is the seventh finding identified in the study.

Finding 7: Sources of evidence point that the architectural hypothesis management positively impacted the product quality.

Evaluating qualitatively the documentary evidence collected from the issues reported after the software was released into production, we observed that none of them was related to the uncertainties represented in the hypotheses. According to the resolution time, none of them required significant effort to be solved. The project was released on the target date, and none of the reported problems prevented the system from being used.

The documentary evidence by itself does not establish a causal relationship between the project’s success and the adoption of ArchHypo. However, that relationship was pointed out directly by five team members, as one of them expressed that its

usage “*reflected directly in the quality and delivery date*” (P1). Another participant emphasized that this quality was combined with other important factors for the client: “*predictability, quality, meeting deadlines, scope, and cost*” (P3). In addition, four respondents also directly mentioned client satisfaction. One justified the relation with the technique, saying that “*a clear vision of what should be prioritized enhances the delivered value*” (P6) and complementing that it “*increased the client satisfaction when using the product*” (P6).

In the process of identifying barriers to the adoption of the technique, the team members unanimously mentioned two obstacles. The first was the need to change the process used previously, which also required a change in culture. Some points highlighted were the introduction of tasks for the technical plans mixed with functional tasks in the project backlog and the introduction of additional activities in the process. These claims lead to the eighth finding.

Finding 8: Changes in the way of working can be an obstacle for professionals to adopt hypothesis management.

A point that was mentioned by several participants was the introduction of architectural tasks in the backlog and the introduction of architectural aspects in some functional tasks. According to them, it was difficult to add “*architectural tasks during the development, which was not practiced by the team in previous projects*” (P5). That change in the project “*was a bit confusing when doing estimates*” (P7), which was confirmed by other participants: “*it was difficult to fit and prioritize the new non-functional tasks in the appropriate moment*” (P2); “*when implementing architectural tasks with development tasks we had problems in estimating*” (P3). According to one of them, that “*overloaded a bit the development team*” (P3), questioning if “*that number*” (referring to architectural tasks) “*could be reduced*” (P3). Changes in the process were also mentioned by two participants as a difficulty: “*the biggest difficulty was identifying and measuring the best moment in the process to introduce new approaches together with organizing the delivery of new packages*” (P2).

Based on some answers, the impression of additional work comes from the fact that some tasks are considered as “*not related to the final product*” (P3). In this case, the team needed to “*relocate resources from development to execute actions from an action plan*” (P8) (referring to the technical plan). That view is created especially by tasks that gather information, such as architectural spikes and software analytics. According to a respondent, “*testing scenarios takes time, so prioritizing the experimentation is hard*” (P4).

The second obstacle, pointed out unanimously by the respondents, is the technique learning curve, which can be hard. Even after understanding how it works, they did not feel ready to apply the technique in another project without support. “*We have a workshop on the use of the technique, but I think it was not enough*” (P7), mentioned one of them. That sentence represents the ninth and last finding well.

Finding 9: *Unanimously, the team members considered the technique hard to learn.*

Some members of the development team mentioned difficulty in “*mapping the scenarios with risk*” (P3, P7, P8) to the hypothesis. “*How to present the assumptions*” (P4) was also considered difficult, referring to the hypothesis specification. “*Definition of the actions*” (P5, P7, P8) to handle the hypothesis was also another point of difficulty raised, as a team member pointed as hard to “*understand the hypotheses as an action driver*” (P5).

Aiming for improvement, one respondent suggested “*provide more training to the team so that we can adhere to the technique correctly*” (P4). This claim is aligned with several others, highlighting that more training would be needed. This participant also suggested including other team members in the process to improve the understanding of the technique, as “*the identification of the hypothesis was initially carried out by the requirements and architecture area*” (P4). Another highlighted that an improvement in that part “*would make the team less dependent on the architecture team*” (P3).

VIII. DISCUSSION

In this section, we discuss our findings, answering our research question “How does using ArchHypo for architectural uncertainty management impact an agile software development project?”. Furthermore, we compare ArchHypo with other approaches existing in the literature and present threats to the study’s validity.

A. ArchHypo Evaluation

According to the DSR approach, one of the goals of the evaluation is to verify if the target artifact is fulfilling its goals and identify improvement points according to the feedback obtained.

Some of the findings extracted from the study results are related and somehow aligned with the ArchHypo goals. From **Finding 2**, we observed that the technique allowed the project to avoid an upfront architectural design. Some of the decisions were made during the iterations after reducing the uncertainty, which allowed better decision-making (**Finding 6**). In other cases, the uncertainties remained until the end of the project but were monitored by the team, as pointed out by **Finding 1**.

An improvement in the development process used by the team was highlighted in **Finding 3**. The team felt more safety in having the uncertainties registered and controlled. This improvement in the process can also have been influenced by the guidelines implemented as part of the technical plans. As indicated by **Finding 5**, that was the strategy most used to handle the hypothesis. However, that was only one technique used in the technical plan. It was not possible to identify any pattern that matches the techniques to the hypothesis quality attributes, and the team considered the context of the project to come up with creative ways to reduce the uncertainty and the impact (**Finding 4**). This finding is aligned with works that

highlight the importance of creativity in software development [47], [48].

Although agile software development practices suggest discussing and adjusting the development process in team retrospectives, we did not find any literature suggesting using process adjustments as a strategy to handle specific issues in the software architecture. In the literature, there are proposals on software development processes that suggest activities focused on a specific quality attribute, such as for safety [54] and usability [55]. However, these works propose processes to be adopted from the beginning and not small changes and adaptations that can tailor the process in response to a specific need. Consequently, observing the frequent utilization of the “guideline implementation” strategy within the project was notable (**Finding 5**). The responses that pointed to improvements in the process also corroborate the success of using such a strategy.

Given all the findings mentioned above, it was expected that ArchHypo would impact the final product and the project’s success, which is represented by **Finding 7**. Although the documentary evidence showed that none of the support requests after the release were related to the hypothesis, we extracted from the answers to the questionnaires that some team members pointed out a causal relationship between the use of ArchHypo and the project’s success.

While Agile methods deal well with functional changes, when these changes are related to the architecture they might affect the whole application, increasing the development cost or provoking huge delays. One of the goals of ArchHypo in working with uncertainties related to the software architecture is to allow an iterative process while avoiding architectural changes with a high impact. One might point out that some of the issues that appeared after the system entered production, such as the problem in data migration and the errors in access control, show that some important uncertainties were not considered in the elicited hypotheses. However, the fact that these issues were solved in a short time shows exactly the opposite: that the impact of these issues were low and, consequently, out of the ArchHypo scope.

The claims that lead to **Finding 8** refer to the change in the way of working as an obstacle, contrasting with others, like **Finding 4** about process improvement. Based on the observations, during the iterations, the team had the impression that adding tasks related to the software architecture “*generated much more work*” (P7) and also identified difficulties in “*including new activities in the process that are not development*” (P8). However, by the end of the project, looking at the overall picture, the team recognized that those additional tasks helped anticipate problems and prepared the team to handle them without the need to postpone the delivery date or work under high pressure. Evidence to support this point is also presented in **Finding 6** and **Finding 7**.

The hard learning curve was also pointed out as an obstacle in **Finding 9**. Considering the characteristics of ArchHypo, it is important to highlight that its adoption requires knowledge of the field of software architecture. For example, to identify uncertainties regarding performance and suggest suitable actions

for the technical plan, it is important to have knowledge and experience in handling that quality attribute. That could also be a reason why training in the technique itself was not considered enough from the perspective of the participants. However, the training can also be extended, including real-life examples and more problem-solving techniques. The technique can also benefit from more guidance in creating the technical plan and from having support tools to manage the hypothesis and execution of the action plan.

As a concrete action that acts on the issues identified, we complemented the patterns used as the basis for ArchHypo with additional patterns that cover practices not yet documented. These patterns covered the basic practices related to hypothesis management [56] and the guidelines that perform small adjustments in the development process to deal with uncertainties [57]. We believe the documentation of such patterns creates a knowledge base that can help practitioners to learn and apply ArchHypo more autonomously.

B. Comparison With Other Approaches

The “Iterative Architecture Approach” proposes to embed architecture design into iterations [17], considering only the necessary architectural features for the current iteration, avoiding predicting and addressing future requirements. A limitation of this approach is that future changes in architecture might have a significant impact. While ArchHypo also proposes to postpone design decisions, the difference is that uncertainty is managed instead of just leaving unknown requirements for future iterations. Both techniques achieve the same result of dividing the architectural features throughout the iterations. However, instead of using the priority of the functional features as a reference and leaving the other ones unhandled, ArchHypo considers the hypothesis assessment and prioritizes actions that prepare for the moment to deal with that architectural feature. In other words, when an architectural decision is delayed, it does not mean that nothing will be done about it since measures to reduce its impact can be implemented. This aspect is not present in the “Iterative Architecture Approach”.

Regarding the division of the architecture work into iterations, another proposal is to create an architecture roadmap that plans its implementation in the time dimension [4]. However, unlike this work, which proposes a more fixed architecture roadmap, ArchHypo considers uncertainty and frequently adapts the plan as the team acquires knowledge.

Several works also point to the flexibility in the architecture as a solution. For example, Hasselbring [18] points to runtime adaptable models and techniques as critical issues for properly integrating architecture work into agile software development. Pang [20], [21] emphasizes the need for a flexible and adaptable software architecture in agile contexts. While implementing flexibility is an important tool to deal with uncertainty, it is not the only one. While these works consider flexibility as the main approach to dealing with future changes in architecture, ArchHypo also adopts other techniques to create the technical plan and handle uncertainty. Flexible solutions are usually more

expensive to develop and might negatively affect other quality attributes.

Some of the earlier foundational approaches in managing architectural concerns are the Architecture Tradeoff Analysis Method (ATAM) [58] and the Architecture-level Modifiability Analysis (ALMA) [59]. While ATAM focuses on assessing consequences of architectural decisions on quality attributes, ALMA specifically focuses on modifiability, targeting one of the following goals: predicting maintenance costs, assessing risks, and comparing architecture alternatives. These techniques can be considered complementary to ArchHypo since they can be part of the technical plan to reduce, but not eliminate, the uncertainty of the impact of an architectural decision in early project phases. ALMA, in particular, can also be used to choose a solution to reduce the impact of future modifications that might be needed due to uncertain factors. Compared to them, ArchHypo has a different goal since it aims to manage the uncertainty during the project iterations. It also relies on other strategies for managing uncertainty, such as delaying an architectural decision while the potential impact is monitored.

Compared to models, such as HYPEX [28] and RIGHT [29], ArchHypo can be considered another experimentation model, similar to the others, i.e., having a similar sequence of steps (identifying hypotheses, design experiments, execute them, and analyze the results). However, it is the first model specifically focused on software architecture, with a particular emphasis on hypotheses, rather than on features and strategic business goals. HYPEX’s initial step is “the generation of features that may potentially bring value to customers and that could be experimented with”. RIGHT emphasizes “assumptions regarding the actions necessary to bring a product or service to market that fulfills the vision and is sustainably profitable”. HyMap, while effective in early-stage startups for identifying functional hypotheses, does not address the architectural uncertainties that ArchHypo is designed to manage. This difference in focus complicates a direct comparison between ArchHypo and the other methods. In addition to that, there are no studies in the literature regarding best practices to handle a system’s architecture using a hypothesis-driven design.

ArchHypo considers, to a different extent, all the steps identified for hypotheses engineering [60]: elicitation, prioritization, specification, analysis, and management. In ArchHypo, the elicitation occurs in the team meeting at the start of the project, but can continue through the iterations. Impact and level of uncertainty, which can be determined by an analysis process, provide an approach for prioritizing hypotheses. For specification, ArchHypo did not enforce any particular template for spelling out hypotheses and this aspect was mentioned as a difficulty in the process by the team members. In this regard, future work could explore whether specific templates could improve the results. Using a specific spreadsheet for listing and updating the status of the hypotheses throughout the project maps to the management activity of hypotheses engineering.

Another related concept is architectural assumption, which is defined as “architectural knowledge taken for granted, or accepted as true without evidence” [61]. These assumptions

are frequently made by software architects [61] but are generally left implicit [62], and a systematic approach for handling them is lacking [61]. ArchHypo is aligned with the view that implicit assumptions should be made explicit, and it proposes doing that through hypotheses, which are explicit and testable. Based on the architectural assumptions literature, ArchHypo also focuses on a broader scope, including any uncertainty that might somehow influence the software architecture, proposing a concrete way to manage and handle hypotheses, representing a contribution that advances the knowledge in the field.

C. Threats to Validity

To discuss potential threats to the validity of our study and how we mitigated them, we followed the suggestions of Runeson and Höst [50] and considered a scheme composed of construct validity, internal validity, external validity, and reliability. We are aware that some qualitative studies in software engineering, such as [63] and [64], have used another scheme composed of transferability, dependability, credibility, and confirmability, which is associated with a constructive or interpretivist research paradigm [65]. However, as usually done in software engineering studies [66], especially given the viewpoint of DSR studies [67], we adopted a pragmatic paradigm, being “concerned with action and change and interplay between knowledge action” [68], which is based on symbolic realism [68] that advocates the existence of a single reality. Therefore, we deemed it more adequate to use the scheme suggested by Runeson and Höst for this study.

Construct validity reflects to what extent the operational measures under study actually represent the intentions of the researcher [50]. A potential issue is, for example, whether researchers and interviewees have different understandings of concepts in interview questions [50]. In this regard, a potential threat to our study was the questionnaire that the participants answered at the end of the project and whether they understood the questions as we expected. This threat is minimal given that the group was cohesive and one of the authors acted as a participant observer who could capture the vocabulary used by the participants. Another potential issue is the uncertainty level and impact assessment, which consisted of scales comprised of three possible values. This threat is minimal because the number of options is small, which could facilitate the participants’ decision, and also because this decision is made by a consensus.

Internal validity concerns causal relations and is threatened when the cause of one factor is attributed to a second factor, but the real cause is a third factor that was not considered [50]. To mitigate this threat, we considered multiple sources of data, i.e., data triangulation. Also, we, as researchers, held regular meetings to discuss the conclusions obtained from the data analysis. A specific threat in this regard is related to the correct application of the technique since the participants had just a one-day training workshop and pointed to the learning curve as a drawback. To mitigate that threat, a participant-observer supported the correct application of the technique during the project. In the analysis process, the researchers confirmed using the documentary evidence, such as in the hypothesis

information and planned tasks, that ArchHypo was applied as expected.

External validity regards the generalization of the findings and to what extent the results are interesting for others outside the cases [50]. In this regard, studies that focus on one project do not rely on statistical generalization as experiments but rather on analytical generalization [49], [50], i.e., “the results are extended to cases that have common characteristics [to the studied case]” [50]. For our study, the case could be considered typical, cf. [49], since it is a software development company developing a bespoke system. However, we acknowledge that the case particularities, such as the development platform and technologies, might influence the results. This limitation is common in DSR studies, as stated by Hevner et al. [36]: “the use of a design artifact on a single project may not generalize to different environments.” Therefore, we believe that further studies in other sites could improve this aspect of ArchHypo.

Reliability concerns to which extent the results are dependent on the specific researchers [50]. A potential threat is the influence of the researcher on the team. We mitigated this threat by following a systematic approach for the method implementation, data collection, and analysis. Another possible issue is the influence of the participant-observer, especially in the training workshop. To mitigate this issue, we limited our influence on the participants, answering only their questions and guiding them according to the practice description.

IX. CONCLUSION

This paper proposed ArchHypo, a technique for managing architectural uncertainty integrated into the software development process. It was evaluated in a mission-critical project to identify its impact on a software development, focusing on its influence on architectural decision-making and the improvement of the software development process. Our analysis indicated that ArchHypo has the potential to play an important role in handling uncertainties, particularly those related to incomplete information regarding system components and requirements, by executing activities to reduce uncertainty and its respective impact. The results of the evaluation provide evidence regarding the technique’s contribution to the successful, on-time release of the project. Notably, 50% of the hypotheses with the highest levels of uncertainty and impact were addressed in the first half of the project, illustrating ArchHypo’s support in prioritizing and managing critical uncertainties. Conversely, it also provides guidance in the management of uncertainties associated with the impact of development activities on some quality attributes, such as usability and quality, which tend to persist until the project’s conclusion. This contrast highlights ArchHypo’s capacity for both resolving specific uncertainties and facilitating the ongoing management of others, thereby improving architectural decision-making processes and the agility of project responses.

Furthermore, the use of ArchHypo provided a systematic approach for allocating and scheduling architectural tasks throughout project iterations, as shown by the execution of

41 activities related to the hypotheses' technical plan, including those requiring client interaction by the Product Owner. The ArchHypo's support to task prioritization and planning, underpinned by hypothesis-driven management, optimizes task allocation and strengthens the overall project management framework, providing a strategy for navigating architectural uncertainties. Feedback from team members highlighted that using ArchHypo "reflected directly in the quality and delivery date," confirming its impact on project outcomes. Its contributions to decision-making processes and process efficiency underscore the potential for further research on its application across different organizational and project contexts. Moreover, investigating tools and methodologies to improve the implementation of ArchHypo represents an important direction for research, potentially expanding its applicability to a larger spectrum of development teams and organizations.

Future work can evaluate the adoption of ArchHypo in companies and development teams with different backgrounds and characteristics. These studies can use data collected prior to the introduction of the technique to be used as a baseline for its evaluation. To evaluate the acceptance of the technique, the UTAUT model [69] could be used in the design of future interview and survey studies. In addition, tools that provide more guidance on using the technique can be developed to reduce its learning curve. For instance, plugins of existing tools, such as Jira and Trello, could introduce support for hypotheses management. Finally, the introduction of guidelines into the activities of the development process, which was a valuable strategy in this project, has not received much support in the literature. Therefore, future studies can also investigate how these guidelines can be adopted as an approach to dealing with uncertainty.

REFERENCES

- [1] L. Viviani, E. Guerra, J. Melegati, and X. Wang, "An empirical study about the instability and uncertainty of non-functional requirements," in *Proc. Agile Processes Softw. Eng. Extreme Program.: 24th Int. Conf. Agile Softw. Develop., XP*, Amsterdam, Netherlands, Jun. 2023, vol. 475, Springer Nature, 2023, p. 77.
- [2] Z. Dragičević and S. Bošnjak, "Agile architecture in the digital era: Trends and practices," *Strategic Manage.*, vol. 24, no. 2, pp. 12–33, 2019.
- [3] E. Woods, M. Erder, and P. Pureur, *Continuous Architecture in Practice: Software Architecture in the Age of Agility and DevOps*. Reading, MA, USA: Addison-Wesley Professional, 2021.
- [4] E. Poort, C. Pautasso, and O. Zimmermann, "Just enough anticipation: Architect your time dimension," *IEEE Softw.*, vol. 33, no. 6, pp. 11–15, Nov./Dec. 2016.
- [5] E. Guerra, R. Wirfs-Brock, and J. Yoder, "Patterns for initial architectural design on agile projects," in *Proc. 4th Asian Conf. Pattern Languages Programs (AsianPLOP)*, 2015, pp. 1–12.
- [6] R. Wirfs-Brock, J. Yoder, and E. Guerra, "Patterns to develop and evolve architecture during an agile software project," in *Proc. 22nd Conf. Pattern Lang. Programs*, 2015, pp. 1–18.
- [7] K. Silva, J. Melegati, X. Wang, M. Ferreira, and E. Guerra, "Using hypotheses to manage technical uncertainty and architecture evolution in a software start-up," *IEEE Softw.*, vol. 41, no. 4, pp. 7–13, Jul./Aug. 2024.
- [8] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. Berlin, Heidelberg: Springer, 2014.
- [9] K. Beck, *Test-Driven Development: By Example*. Reading, MA, USA: Addison-Wesley, 2003.
- [10] E. Guerra and M. Aniche, "Achieving quality on software design through test-driven development," in *Software Quality Assurance*. San Mateo, CA, USA: Morgan Kaufmann, 2016, pp. 201–220.
- [11] G. Bavota, A. D. Lucia, M. D. Penta, R. Oliveto, and F. Palomba, "An experimental investigation on the innate relationship between quality and refactoring," *J. Syst. Softw.*, vol. 107, pp. 1–14, Sep. 2015.
- [12] N. Ford, "Evolutionary architecture and emergent design: Environmental considerations for design, part 2," [Online]. Available: <https://www.ibm.com/developerworks/java/library/j-eaed18/index.html#24>
- [13] M. Waterman, J. Noble, and G. Allan, "How much up-front? A grounded theory of agile architecture," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, vol. 1, Piscataway, NJ, USA: IEEE Press, 2015, pp. 347–357.
- [14] H. P. Breivold et al., "What does research say about agile and architecture?" in *Proc. 5th Int. Conf. Softw. Eng. Adv.*, Piscataway, NJ, USA: IEEE Press, 2010, pp. 32–37.
- [15] N. Ford "Evolutionary architecture and emergent design: Environmental considerations for design, part 2," 2020. [Online]. Available: <https://www.ibm.com/developerworks/java/library/j-eaed18/index.html#24>
- [16] U. Zdun, R. Capilla, H. Tran, and O. Zimmermann, "Sustainable architectural design decisions," *IEEE Softw.*, vol. 30, no. 6, pp. 46–53, Nov./Dec. 2013.
- [17] C. Yang, P. Liang, and P. Avgeriou, "A systematic mapping study on the combination of software architecture and agile development," *J. Syst. Softw.*, vol. 111, pp. 157–184, Jan. 2016.
- [18] W. Hasselbring, "Software architecture: Past, present, future," in *Essence of Software Engineering*, Cham, Switzerland: Springer, 2018, pp. 169–184.
- [19] D. Yong, "Design and practice of software architecture in agile development," *J. Phys.: Conf. Ser.*, vol. 2074, no. 1, Nov. 2021, Art. no. 012008.
- [20] C.-Y. Pang, *Applying Software Engineering Design Principles to Agile Architecture*. Hershey, PA: IGI Global, 2020, pp. 82–108.
- [21] C.-Y. Pang, *Applying Software Engineering Design Principles to Agile Architecture*. Hershey, PA: IGI Global, 2022, pp. 330–355.
- [22] M. Erder and P. Pureur, *Continuous Architecture: Sustainable Architecture in an Agile and Cloud-Centric World*. San Mateo, CA, USA: Morgan Kaufmann, 2015.
- [23] S. Munari, S. Valle, and T. Vardanega, "Microservice-based agile architectures: An opportunity for specialized niche technologies," in *Proc. Ada-Europe Int. Conf. Reliable Softw. Technol.*, Cham, Switzerland: Springer, 2018.
- [24] D. Taibi, V. Lenarduzzi, and C. Pahl, "Continuous architecting with microservices and devops: A systematic mapping study," in *Proc. Int. Conf. Cloud Comput. Services Sci.*, Cham, Switzerland: Springer, 2018.
- [25] B. Holmes and A. Nicolaescu, "Continuous architecting: Just another buzzword," *Full-Scale Software Engineering the Art of Software Testing*, pp. 1–6, 2017.
- [26] A. Martini and J. Bosch, "A multiple case study of continuous architecting in large agile companies: Current gaps and the caffeine framework," in *Proc. 13th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA)*, 2016, pp. 1–10.
- [27] E. Lindgren and J. Münch, "Raising the odds of success: The current state of experimentation in product development," *Inf. Softw. Technol.*, vol. 77, pp. 80–91, 2016.
- [28] H. H. Olsson and J. Bosch, "From opinions to data-driven software R & D: A multi-case study on how to close the 'open loop' problem," in *Proc. 40th EUROMICRO Conf. Softw. Eng. Adv. Appl.*, Piscataway, NJ, USA: IEEE Press, 2014.
- [29] F. Fagerholm et al., "The RIGHT model for continuous experimentation," *J. Syst. Softw.*, vol. 123, pp. 292–305, Jan. 2017.
- [30] J. Melegati, X. Wang, and P. Abrahamsson, "Hypotheses engineering: First essential steps of experiment-driven software development," in *Proc. IEEE/ACM Joint 4th Int. Workshop Rapid Continuous Softw. Eng. 1st Int. Workshop Data-Driven Decisions, Experimentation Evolution (RCoSE/DDrEE)*, Piscataway, NJ, USA: IEEE Press, 2019.
- [31] J. Melegati, E. Guerra, and X. Wang, "Understanding hypotheses engineering in software startups through a gray literature review," *Inf. Softw. Technol.*, vol. 133, May 2021, Art. no. 106465.
- [32] J. Melegati, E. Guerra, and X. Wang, "HyMap: Eliciting hypotheses in early-stage software startups using cognitive mapping," *Inf. Softw. Technol.*, vol. 144, Apr. 2022, Art. no. 106807.
- [33] R. Baskerville, "What design science is not," *Eur. J. Inf. Syst.*, vol. 17, no. 5, pp. 441–443, Oct. 2008.
- [34] S. Gregor, "Building theory in the sciences of the artificial," in *Proc. 4th Int. Conf. Des. Sci. Res. Inf. Syst. Technol.*, 2009, pp. 1–10.
- [35] S. Gregor and A. R. Hevner, "Positioning and presenting design science research for maximum impact," *MIS Quart.*, vol. 37, no. 2, pp. 337–355, Jun. 2013.

- [36] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quart.*, vol. 28, no. 1, pp. 75–105, Mar. 2004.
- [37] J. Melegati and X. Wang, "Quest: new practices to represent hypotheses in experiment-driven software development," in *Proc. 2nd ACM SIGSOFT Int. Workshop Softw.-Intensive Bus.: Start-ups, Platforms, Ecosyst.*, 2019, pp. 13–18.
- [38] M. Glinz, "A risk-based, value-oriented approach to quality requirements," *IEEE Softw.*, vol. 25, no. 2, pp. 34–41, Mar./Apr. 2008.
- [39] E. Lindgren and J. Münch, "Raising the odds of success: The current state of experimentation in product development," *Inf. Softw. Technol.*, vol. 77, pp. 80–91, Sep. 2016.
- [40] M. Kim, T. Zimmermann, R. DeLine, and A. Begel, "The emerging role of data scientists on software development teams," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 96–107.
- [41] O. Baysal, R. Holmes, and M. W. Godfrey, "Developer dashboards: The need for qualitative analytics," *IEEE Softw.*, vol. 30, no. 4, pp. 46–52, Jul./Aug. 2013.
- [42] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Reading, MA, USA: Addison-Wesley, 2004.
- [43] E. Guerra, E. Buarque, C. Fernandes, and F. Silveira, "A flexible model for crosscutting metadata-based frameworks," in *Proc. Comput. Sci. Its Appl.–ICCSA 2013: 13th Int. Conf.*, Ho Chi Minh City, Vietnam, 2013, Berlin, Heidelberg: Springer, 2013, pp. 391–407.
- [44] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: State of the art and research challenges," *IEEE Trans. Services Comput.*, vol. 11, no. 2, pp. 430–447, Mar./Apr. 2017.
- [45] S. Mahdavi-Hezavehi, V. H. Durelli, D. Weyns, and P. Avgeriou, "A systematic literature review on methods that handle multiple quality attributes in architecture-based self-adaptive systems," *Inf. Softw. Technol.*, vol. 90, pp. 1–26, Oct. 2017.
- [46] J. W. Yoder and R. Wirfs-Brock, "QA to AQ part two: Shifting from quality assurance to agile quality: Measuring and monitoring quality," in *Proc. 21st Conf. Pattern Lang. Programs*, 2014, pp. 1–20.
- [47] R. Gutbrod and C. Wiele, *The Software Dilemma: Balancing Creativity and Control on the Path to Sustainable Software*. Berlin, Heidelberg: Springer Science & Business Media, 2012.
- [48] R. Mohanani, P. Ram, A. Lasisi, P. Ralph, and B. Turhan, "Perceptions of creativity in software engineering research and practice," in *Proc. 43rd Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, 2017, pp. 210–217.
- [49] R. Yin, *Case Study Research: Design and Methods*, ser. Applied Social Research Methods. Newbury Park, CA, USA: Sage, 2003.
- [50] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Eng.*, vol. 14, no. 2, pp. 131–164, 2009.
- [51] C. Wohlin and A. Rainer, "Is it a case study?—A critical analysis and guidance," *J. Syst. Softw.*, vol. 192, 2022, Art. no. 111395.
- [52] T. C. Lethbridge, S. E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," *Empirical Softw. Eng.*, vol. 10, no. 3, pp. 311–341, 2005.
- [53] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Res. Psychol.*, vol. 3, no. 2, pp. 77–101, 2006.
- [54] G. Islam and T. Storer, "A case study of agile software development for safety-critical systems projects," *Rel. Eng. & Syst. Saf.*, vol. 200, 2020, Art. no. 106954.
- [55] F. A. Salman and A. Deraman, "A model for incorporating suitable methods of usability evaluation into agile software development," *Bull. Elect. Eng. Inform.*, vol. 11, no. 6, pp. 3433–3440, 2022.
- [56] K. Silva, L. Adolfo, A. Coppe, F. Silveira, M. Ferreira, and E. Guerra, "Patterns for using hypothesis engineering to manage architectural uncertainties," in *Proc. 29th Eur. Conf. Pattern Lang. Programs*, 2024, pp. 1–8.
- [57] A. Paris, E. Guerra, F. Silveira, and K. Silva, "Patterns for small adjustments in the development process to deal with architectural uncertainties," in *Proc. 29th Eur. Conf. Pattern Lang. Programs*, 2024, pp. 1–8.
- [58] R. Kazman, M. Klein, and P. Clements, *ATAM: Method for Architecture Evaluation*. Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA, USA, 2000.
- [59] P. Bengtsson, N. Lassing, J. Bosch, and H. van Vliet, "Architecture-level modifiability analysis (ALMA)," *J. Syst. Softw.*, vol. 69, no. 1–2, pp. 129–147, Jan. 2004.
- [60] J. Melegati, E. Guerra, and X. Wang, "Understanding hypotheses engineering in software startups through a gray literature review," *Inf. Softw. Technol.*, vol. 133, 2021, Art. no. 106465.
- [61] C. Yang, P. Liang, P. Avgeriou, U. Eliasson, R. Heldal, and P. Pelliccione, "Architectural assumptions and their management in industry – An exploratory study," in *Proc. 11th Eur. Conf. Softw. Architect.* vol. 10475, 2017, pp. 191–207.
- [62] C. Yang, P. Liang, and P. Avgeriou, "A survey on software architectural assumptions," *J. Syst. Softw.*, vol. 113, pp. 362–380, Mar. 2016.
- [63] K.-J. Stol, P. Avgeriou, M. A. Babar, Y. Lucas, and B. Fitzgerald, "Key factors for adopting inner source," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 2, 2014, pp. 1–35.
- [64] W. Hussain et al., "How can human values be addressed in agile methods a case study on SAFe," *IEEE Trans. Softw. Eng.*, vol. 48, no. 12, pp. 1–1, Dec. 2022.
- [65] E. G. Guba and Y. S. Lincoln, "Competing paradigms in qualitative research," in *Handbook of Qualitative Res.*, Thousand Oaks: SAGE Publications, 1994, pp. 163–194.
- [66] J. Melegati and X. Wang, "Surfacing paradigms underneath research on human and social aspects of software engineering," in *Proc. IEEE/ACM 13th Int. Workshop Cooperative Human Aspects Softw. Eng. (CHASE)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 41–50.
- [67] P. Runeson, E. Engström, and M.-A. Storey, "The design science paradigm as a frame for empirical software engineering," in *Contemporary Empirical Methods in Software Engineering*, Cham, Switzerland: Springer International Publishing, 2020, pp. 127–147.
- [68] G. Goldkuhl, "Pragmatism vs interpretivism in qualitative information systems research," *Eur. J. Inf. Syst.*, vol. 21, no. 2, pp. 135–146, 2012.
- [69] V. Venkatesh, M. G. Morris, G. B. Davis, and F. D. Davis, "User acceptance of information technology: Toward a unified view," *MIS Quart.*, vol. 27, no. 3, pp. 425–478, 2003.