Call 2023, Proyectos en Colaboración Público-Privada
Technical description

1. Front matter

Coordinator: OSOCO

Participant: Universidad Rey Juan Carlos

TITLE OF THE PROJECT:
ADVISE: Advanced Vision on Intelligent Software Engineering

Contents

3. Executive summary

In recent years, generative artificial intelligence (AI) models have demonstrated their capabilities to generate source code from informal specifications. This advancement is broadening the horizons for code development and maintenance, profoundly impacting software engineering practices and tools. Specifically, these generative models present an opportunity to bring software production closer to domain experts with limited programming skills, by abstracting some or all of the source code details.

However, several challenges must be addressed to effectively integrate AI systems into software development processes. For instance, these systems can sometimes generate plausible but incorrect code, a phenomenon known as 'hallucination'; ensuring that the AI agents produce the intended code is not straightforward, as crafting the right prompts for complex problems can be challenging; moreover, it is difficult to specify the nuances of a problem in natural language such that the implementation produced by the AI assistant aligns with the human's expectations.

In this project, we propose a framework for integrating AI agents in the processes for software development and maintenance. This framework aims to ensure that the code produced with the assistance of AI assistants satisfies certain functional and non-functional requirements and closely aligns with the intentions of the human specifying the desired functionality. We will explore two approaches: scenario-based modeling (SBM) and specify-test-check loops (STCL). The first one is inspired by the play-in play-out paradigm, while the second one borrows from current practices in large free, open source software (FOSS) projects, and from test-driven development. Both approaches will be used to construct specific implementations of the framework, along with associated proof-of-concept implementations. We will also conduct empirical experiments using these implementations to assess the feasibility and soundness of our solutions, and will carry out real-world case studies to evaluate and validate them.

We anticipate that the project outcomes will pave the way for integrating AI assistants into the production and maintenance of software, including complex software systems. This integration is expected to enhance the competitiveness of the industrial partner in the consortium, and elevate the academic partner's scientific significance. Simultaneously, we foresee significant technology transfer impacts on the national and international software sector, owing to the availability of all the solutions produced as FOSS components, and to an active strategy of leveraging this for a focused marketing strategy.

Keywords: *Generative AI • Software Engineering • Specify-Test-Check Loop (STCL) • Scenario-Based Modeling (SBM) • Test-Driven Development (TDD) • Live Sequence Charts (LSC) • AI-Assisted Code Generation • Model-Driven Development • Behavioral Programming (BP) • Domain-Driven Design (DDD) • End-User Programming • Human Computer Interfaces*

Palabras clave: *Inteligencia Artificial Generativa • Ingeniería de Software • Bucle de Especificar-Probar-Verificar • Modelado Basado en Escenarios • Desarrollo Guiado por Pruebas • Diagramas de Secuencia Vivos • Generación de Código Asistida por IA • Desarrollo Guiado por Modelos • Programación basada en Comportamiento • Diseño Guiado por el Dominio • Programación por el Usuario Final • Interfaces Hombre-Computadora*

4. Goals and expected results

In the rapidly advancing field of computer science, the emergence of deep learning has led to transformative changes. Among these changes, the development of broad-spectrum generative machine learning (ML) models such as ChatGPT [27, 1, 39] and Llama [34] stands out. These tools have immense potential, particularly in the modeling and coding of complex systems, but they also pose unique challenges.

In this project, we aim to harness these models to create tools that automate significant aspects of software development and maintenance processes. To achieve this, we will build upon previous research on how large-scale, free, open-source software projects operate to produce and maintain software, as well as on scenario-based modeling. We have discovered that these two areas, although seemingly unrelated at first glance, are closely intertwined and can be advanced further through the use of generative ML models.

Our ultimate goal is to ascend one rung on the abstraction ladder. In certain scenarios, this would enable domain experts, who may not necessarily be proficient in a programming language, to develop code with zero or near-zero programming. Importantly, they would still be able to ensure that the generated software meets their functional and non-functional requirements. This approach could revolutionize the way we think about and approach software development, opening up new possibilities for efficiency and innovation.

4.1 A Wise Computing Approach for Harnessing Generative ML in Software Engineering

The notion of automated software generation has its origins in the early compilers, which were conceptualized as “automatic code generators”: the introduction of high-level languages marked a significant leap in abstraction compared to assembler languages. Since then, “automatic code generation” has been a holy grail, getting new meanings as the state of the art advanced. One of those approaches, relatively popular in recent times, has been model-driven development [30]. This approach has investigated methods to articulate software system requirements and automatically generate code to fulfill those requirements. However, these methodologies have only achieved success in specific, relatively confined domains.

The emergence of generative AI has instigated a paradigm shift in the field, showcasing its ability to generate diverse types of content [3, 10], and execute tasks pertinent to the creation or maintenance of software systems [17, 7]. For instance, Large Language Models (LLMs) can be employed for automatic code generation from natural language descriptions of requirements [31], for test generation [36], or for bug fixing [19].

Despite their power, these tools often yield results that can be inaccurate or incomplete, potentially neglecting critical aspects of input queries [22], or generating plausible yet incorrect code, low-quality code, or code with security vulnerabilities [33, 12]. Furthermore, the input queries formulated by human engineers might be flawed, leading to compounded inaccuracies in the final models and code. As engineers increasingly depend on these AI tools, the risk of these inaccuracies being incorporated into final products intensifies.

Consequently, there is a pressing need to design processes and tools that leverage the capabilities of LLMs, while circumventing their shortcomings, and ensuring that the software systems constructed with their assistance meet the functional and non-functional requirements defined by domain experts. This project aims to bridge this gap by exploring two different approaches: one based on specify-test-check loops, and the other on extending scenario-based modeling. Both approaches will be tested in a case study of interest to the industrial partner: the production of reactive systems based on microservices.

4.1.1 Proposed approach

Our approach integrates advanced generative AI capabilities with a deep understanding of software engineering complexities. The project aims to address this challenge by proposing a controlled and iterative approach to utilizing tools based on LLMs in the software development cycle. The core idea is to employ these tools for various tasks but then subject their outputs to thorough automatic inspection and analysis, in a process under the control of the humans driving it. This ensures the soundness and accuracy of the results.

A pivotal aspect of this project is its alignment with David Harel's vision of "Wise Computing" [13]. This philosophy envisions transforming the computer into an active, collaborative member of the software engineering team. In this vision, the computer raises questions, offers suggestions and observations, and conducts verification-like processes proactively, even without explicit instructions. This approach is a deliberate counterpoint to the notion of AI simply replacing human software engineers' tasks. Instead, it emphasizes a harmonious and cooperative interaction, where AI complements and enhances human expertise.

The methodology is designed to augment, not replace, human expertise, adhering to the following principles:

1. Iterative Invocation and Verification: Repeatedly invoking LLM-based tools for tasks like scenario generation, code synthesis, and system modeling, followed by a rigorous validation process, which will be automated, but tightly controlled by human engineers.

2. Balancing AI Assistance with Human Oversight: Establishing a workflow where AI tools significantly reduce engineers' workload but do not compromise the quality of the resulting products. Human expertise remains central to the process, guiding and correcting the AI outputs.

3. Development of Verification and Validation Protocols: Creating robust protocols for verifying and validating the outputs of AI tools, ensuring that they meet the high standards required in software engineering.

4. Adhering to 'Wise Computing' Principles: Ensuring AI tools act as collaborative members of the engineering team.

5. Moving towards Zero-Code or Low-Code Software Development and Maintenance: Allowing domain experts to develop and maintain software without having to understand the underlying source code, or at least most of it.

We also draw from domain-driven design (DDD) practices, improving the way teams in charge of developing and maintaining software understand and model complex business domains. Our approach extends the domain discovery process, typically initiated with techniques like event storming, into a more dynamic and iterative modeling practice.

Event storming, as described originally by Alberto Brandolini [2], lays the groundwork for understanding complex business domains. In our approach, the insights gained from event storming sessions are further developed with generative AI helping to translate natural language descriptions into operational scenarios. This method aligns closely with the principles outlined in Eric Evans' seminal work on domain-driven design [9], particularly in its emphasis on a ubiquitous language and a shared understanding of the domain.

Our approach is not to provide tools just as a means for domain discovery, but also as a platform for visualizing and validating the operational behavior of the system. This way, stakeholders, both technical and non-technical, are enabled to "see" the system in action and understand how business scenarios play out in real-time. This aspect is crucial for verifying that the system's behavior aligns with business expectations and rules, which emphasizes the importance of a model-driven approach in aligning software design with business needs [35].

Therefore, we offer a novel and effective toolset for continuous domain discovery and validation, providing a bridge between the initial conceptual understanding of a domain and its operational realization, ensuring that the final software product truly reflects the complexities and nuances of the business domain it is designed to serve.

4.1.2 Objectives

The main research objective of this project can be described as follows:

> "To design, test and verify a software production and maintenance process assisted by generative AI, capable of producing software systems fulfilling functional and non-functional specifications, following automatic procedures guided by human domain experts."

The main outcome of the project, given this objective, will be:

> "A software system and associated toolset capable of working with domain experts to incrementally define the specifications of a software system, producing such system in a way that the expert can check, assisted by the tool, its functional and non-functional characteristics."

We are aware that maybe we cannot fully achieve the intended goal (see Section 5.4, needing to complement automatic procedures with manual assistance by humans. But even if we can only partially achieve it, the resulting outcome will be a large incremental leap with respect to the current procedures for developing and maintaining software in most domains.

To minimize the risk of not achieving the main objective, we will try two different research approaches: one based in the already mentioned specify-test-check loop (STCL), and the other one based on extending scenario-based modeling (SBM). Both will be implemented as a tool for a specific domain of interest to the industrial partner: building microservices-based reactive applications. For that implementation, we will use a common toolkit that will make cost-effective to build both tools, thanks to the reuse of many modules that both approaches will have in common. Finally, both tools will be tested in the production of real systems of interest of the industrial partner.

Therefore, to achieve the intended main objective, we have identified the following intermediate objectives:

O-Research-SBM. Produce a sound software development and maintenance process, assisted by generative AI, following the scenario-based modeling (SBM) approach.

O-Research-STCL. Produce a sound software development and maintenance process, assisted by generative AI, following the specify-test-check loop (STCL) approach.

O-Tech-Toolkit. Common toolkit, with modules implementing functionality common to both approaches.

O-Tool-SBM. Tool for implementing the SBM approach for building microservices-based reactive applications.

O-Tool-STCL. Tool for implementing the STCL approach for building microservices-based reactive applications.

O-Validation. Validation of the designed processes and the implemented tools in real use cases.

In the rest of this chapter, both research approaches, the toolkit and tools, and the validation use cases will be introduced in more detail.

4.2 ML-assisted scenario-based modeling

This activity focuses on enhancing Scenario-Based Modeling (SBM) [14, 16] with generative AI to interpret modelers' intentions in natural language and use them to "play in" [15] behavior scenarios.

The resulting tool aims to make scenario-based programming more intuitive and accessible, particularly for non-experts.

4.2.1 Scenario-based modeling overview

When we consider the natural way humans approach understanding complex systems, we often find ourselves thinking in scenarios rather than dissecting each component's behavior. For instance, consider a person navigating a smart home system. Rarely would you hear someone describe the system in terms of its individual states, such as "The thermostat is set to heating mode, the lights are in energy-saving mode, and the security system is armed.". Instead, a more intuitive description might be scenario-based, like: "When I arrive home in the evening, I want the house to be warm, the lights to gradually brighten, and the security system

to disarm as soon as my phone connects to the home Wi-Fi.". This approach encapsulates a series of interactions and responses between various components of the smart home system, portraying a complete scenario that aligns more naturally with human cognition.

In this context, Scenario-Based Modeling (SBM) emerges as an alternative and more intuitive approach to traditional programming. While conventional methods often focus on detailed descriptions of the internal states and behaviors of each object or system component (an "intra-object" approach), SBM shifts towards understanding and designing global interactions and behaviors (an "inter-object" approach). In this new *behavioral programming* (BP) paradigm the user specifies the system behavior in an incremental way by specifying independent scenarios. Figure 4.1 visualizes these two approaches for specifying behavior.

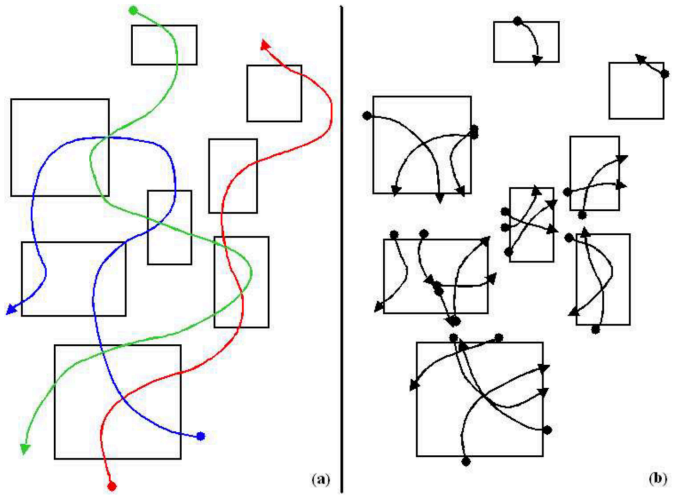


Figure 4.1: (a) Inter-object vs. (b) intra-object behavior (from [14])

This methodology leverages the human tendency to conceptualize complex systems through scenarios and narratives that describe how different elements interact with each other to achieve an outcome or respond to a specific stimulus. By focusing on how objects cooperate and participate in broader scenarios, SBM facilitates the creation of software models that are more accessible and understandable for developers and other stakeholders, including those without deep technical expertise in programming. This approach not only enhances communication and understanding within development teams but also fosters a more holistic and cohesive view of the system's behavior.

In addition to the aforementioned advantage, Scenario-Based Modeling (SBM) also addresses a fundamental issue in conventional software system development: traditionally, the set of requirements is not considered an executable model of the system. As a result, various "soft" methodological recommendations, i.e., non-automatable guidelines, have been proposed to informally lead developers in constructing the system model and, subsequently, its implementation in code. Even in approaches advocating domain understanding and model creation, such as Domain-Driven Design (DDD), there exists a gap between requirements, models, and executable code that designers must navigate with the aid of manual, heuristic-based processes, principles, and patterns. The figure 4.2 depicts this traditional system development cycle where the dashed lines represents these "soft" processes.

The efforts to bridge this gap between the way people form and express their thoughts about the system under development and the possible formats these thoughts can take that can eventually be executed by a computer, give rise to multiple problems like the following:

Correctness. The translation from conceptual to code is costly and error-prone, so there are few guarantees that, even if the requirements are correct and meet our expectations, the system model and the executable code will be equally correct.

Comprehension. The initial understanding of the expected behavior becomes obscured as we evolve towards executable code, becoming so specialized that it is no longer accessible to non-technical participants.

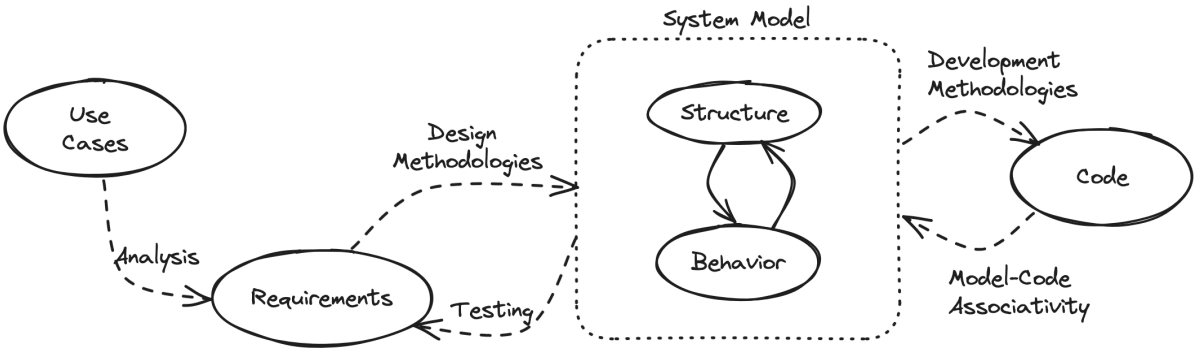


Figure 4.2: Conventional software development lifecycle

Requirements-Code Alignment. It becomes challenging to associate requirements with code, and vice versa, as the behaviors are typically "scattered" across the code structure, complicating the system's future maintenance.

SBM seeks to offer a solution to this problem by considering scenario-based behavior as genuinely executable. Instead of informally guiding designers to build the system, the idea is to automatically synthesize an implementation directly from these scenarios. This approach ensures a more seamless transition from the problem space (requirements) to the solution space (implementation), maintaining the clarity and integrity of the original requirements throughout the development process. It also allows for greater involvement of non-technical stakeholders, as the scenarios remain comprehensible and directly related to the implemented behavior. Most importantly, it facilitates the maintenance of the system, as the direct correlation between requirements and code is preserved, making it easier to track changes and updates in line with evolving business needs.

How it works

The SBM approach involves a visual language to express scenario-based behavior, known as Live Sequence Charts (LSCs) [6]. The visual language of LSCs allows specifying scenarios of what may happen, what must happen and what must not happen. These scenarios are based on classical sequence diagrams (MSCs) with the additional modalities of must/may/forbid and they can be executed directly.

The LSC language has been extended with a tool (the *Play-Engine*) that supports intuitive GUI-based methods for capturing the behavior (termed *play-in*) and for executing a set of LSCs (termed *play-out*) [15].

The main idea of the play-in process is to raise the level of abstraction in requirements engineering, and to work with a look-alike version of the system under development. This enables people who are unfamiliar with LSCs, or who do not want to work with such formal languages directly, to specify the behavioral requirements of the system using a high-level, intuitive and user-friendly mechanism.

With play-in the system's designer first builds the GUI of the system, with no behavior built into it, with only the basic methods supported by each GUI object. The user then 'plays' the incoming events on the GUI, by clicking buttons and rotating knobs in an intuitive drag & drop manner. By similarly playing the GUI, often using right-clicks, the user then describes the desired reactions of the system and the conditions that may o must hold. As this is being done, the Play-Engine tool constructs the corresponding LSCs automatically.

After playing in a part of the behavior, the natural thing to do is to make sure that it reflects what the user intended to say. Instead of doing this the conventional way, by building an intra-object model, or prototype implementation, we would like to test the inter-object behavior directly. Accordingly, we extend the power of our GUI-intensive play methodology, to make it possible not only to specify and capture the required behavior but to test and validate it as well. And here is where the complementary play-out mechanism enters. In play-out, the user simply

plays the GUI application as he/she would have done when executing a system model, or the final system.

The LSC specification, together with the play-in/play-out approach, may be considered to be not just the system's requirements but actually its final implementation. This would represent a paradigm shift in the software development life cycle.

4.2.2 Smart play-in

This project aims to enhance the conventional play-in methodology by introducing an enriched play-in method. This method will create a more advanced interface for defining system requirements and for scenario-based programming.

Our approach is to develop a modeling environment that incorporates an intelligent user interface. This interface will blend GUI-based methods with natural language processing capabilities. Users defining the system's behavior will have the flexibility to choose the most suitable method for their modeling skills or the specific type of behavior they are working with. They can interact with the GUI representation of the system or describe the scenario in natural language. Our enhanced *Smart Play-Engine* tool will integrate generative AI into the traditional play-in process, enabling it to interpret the user's intentions and provide guidance during design sessions.

While conventional play-in is user-friendly and easily adoptable by individuals without a programming background, it is not without its limitations. One such limitation is the need for a pre-prepared GUI of the system, which is still non-functional. Here, generative AI can assist users by automatically and iteratively generating this mock-up GUI.

Additionally, many requirements are inherently less interactive and more programmatic, such as specifying conditions, loops or selecting variables. In these instances, our 'Smart Play-In' feature becomes particularly valuable. Natural language descriptions are quick and straightforward, and they can be seamlessly integrated with GUI interactions.

This AI-enhanced process is designed to make scenario-based programming more accessible and intuitive, especially for those lacking extensive experience in formal programming languages. The development process is going to be more a 2-way conversation between designers and AI agents, where the program's behavior is written in collaboration. This view contrasts with the view of a "program as a sequence of instructions to be obeyed".

The figure 4.3 shows the potential shift we envision in the software development lifecycle (SDLC) if this *behavioral and conversational programming* approach were adopted.

4.2.3 Some remarks

By merging the 'Play-In' and 'Play-Out' processes, our goal is to develop a comprehensive modeling environment where AI serves as a design assistant. This tool will not only aid in specifying scenarios but also assist in their dynamic execution and testing, ensuring a robust and user-friendly modeling environment.

Ultimately, this innovative approach is expected to transform scenario-based programming, making it more inclusive, efficient, and in tune with human cognitive processes.

The work presented here builds upon the original play-in idea of Harel and Marelly [15], which allows user interaction with a GUI for specifying behavior. However, user interaction for capturing behavior is also found in many *Programming by Demonstration* systems since 1975 like the Pygmalion, Cocoa or Stagecase environments [5].

By the other way, the idea of multimodal interfaces is discussed by Ingebretsen [18] and it is already in use in intelligent interfaces for gaming and in smart phones. These modalities include speech, facial expression, body posture, gestures and bio-signals.

In summary, this approach allows an alternative way of programming the behavior of a reactive system, which is totally scenario-based and inter-object in nature. Basic to this is the idea that LSCs can actually constitute the implementation of a system, with the play-out algorithms and the Play-Engine being a sort of 'universal reactive machine' that executes the LSCs instead of a conventional implementation, as opposed to the dominant Code-centric Development (CcD).

If developers and end-users adopt this view, behavioral specification of a reactive system would not have to involve any intra-object modeling or coding task.

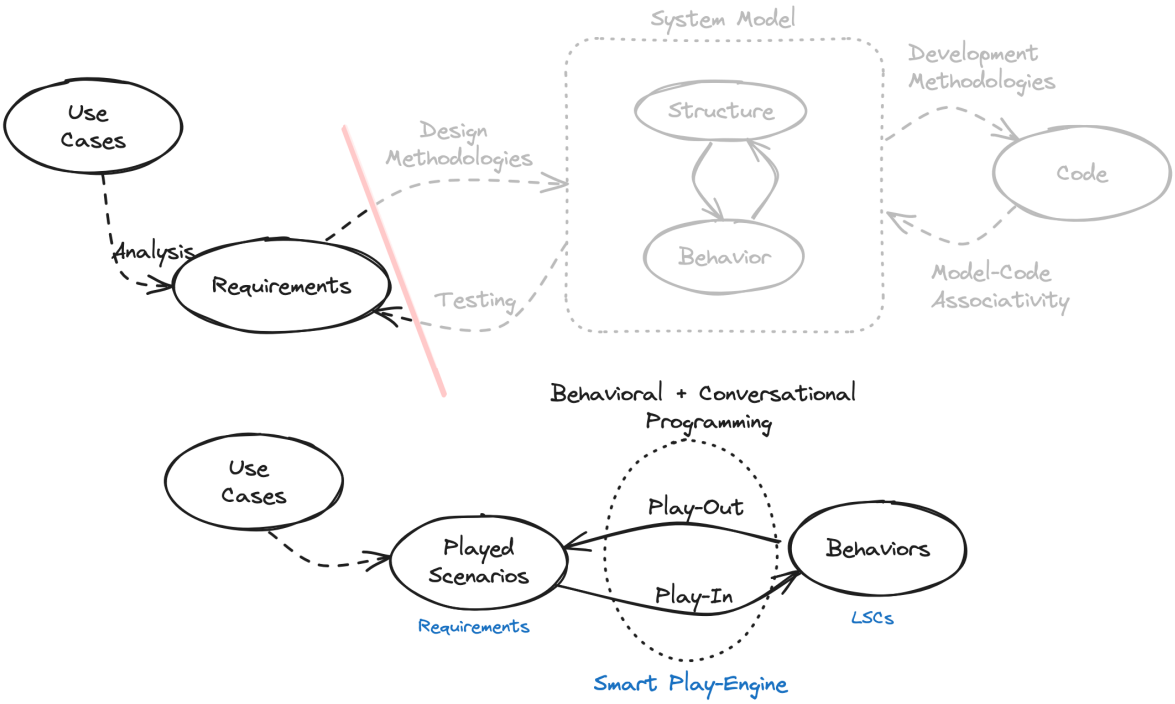


Figure 4.3: Changes in the SDLC using behavioral and conversational programming

Moreover, the advantages of having the system's behavior structured according to the way the engineers invent and design it and the users comprehend it will be very relevant, for example, in the testing, maintenance and modifications stages, or in sharing the specification with less technically oriented people.

4.3 ML-assisted specify-test-check loop

We propose a model that mirrors the way reliability and code quality are improved in many large free and open source software (FOSS) projects. In these projects, changes to the source code undergo a well-established process before acceptance. Developers fix bugs, enhance the software, or add new functionality by producing patches, which are thoroughly reviewed, discussed, and then accepted into the code base. This process may require several iterations over the proposed patch. Moreover, it is increasingly common to have detailed testing policies, which usually require tests to be included in the patch, and regression and integration tests to be passed before considering acceptance [24]. This leads to a natural division of tasks in the project, with most developers devoted to writing changes to the code and tests for it, while a small number of more experienced developers tend to devote most of their time to review. Reviewers may be assisted by automatic tools for code analysis and testing, which let them focus on changes that are worth checking.

We envision an evolution of this model with more generative AI in the loop. Code changes could be produced more automatically, with humans focusing on describing what the code should do and reviewing the produced code. Tests could also be built automatically, and other tools could help ensure that the code meets some quality standards. Ultimately, humans could just explain to generative AI agents how they want the code to perform, so that agents produce the code and comprehensive sets of tests. Humans would then check that the testing cases are complete enough, and that they pass before considering accepting the patch. This model has already been partially tried [11].

A natural progression in this field could enable the production and maintenance of complex software systems without humans having to interact with source code at all. This can be achieved by applying the previous discussion to zero-code development. Generative models are already proving their ability to produce, with some reliability, small code fragments following specifications. We can explore how to improve this process by using tests, and how to let

domain experts drive the production of the code without actually interacting with the code. For combining these code fragments into complete systems, we can explore the use of XR-based graphical interfaces.

4.3.1 General loop

The production of small pieces of code, such as functions, could start with a domain expert specifying, in natural language, how the code should perform. Depending on the capabilities of the model, that specification could also include hints for the implementation, such as which algorithm to use. Then, the process for producing the code is based on two incremental loops, both assisted by generative models:

- Test Production: The specification produced by the expert is used as input to a generative model, which is instructed to produce a collection of tests for it. Since the collection could be incomplete or include erroneous tests, the generative model will be asked to produce an explanation of the tests, including the input and the expected output for each of them. The expert will review that explanation, asking for completion or changes to the test suite, which the generative model will produce. The process will continue, with the expert refining the tests until they are considered valid. The process may include techniques such as smart mutation testing [29], so that the system can also improve tests, and suggest new tests, in principle not foreseen by the human, which could check for corner situations, of being the result of not previously known constraints.
- Code Production: Once the test suite is adequate, the generative model is asked to produce the piece of code using the test suite and the specification as input, and a second refinement cycle starts. This one is completely automated: the test suite is run on the produced code, and if any test fails, the output of the execution, along with the specification and the test suite is fed to the generative model, until all tests pass. In this loop, automated analysis tools can also be used to ensure that the produced code meets some constraints, for example with respect to complexity, performance, quality, avoidance of bugs detected with static analysis, etc. The loop can also produce, in each iteration, a number of solutions that are ranked according to some metric, so that the human has control not only on functional, but also on non functional measurable requirements.

This way, the final version of the code will be compliant with the test suite. If the expert has training on how to build good test suites, and good prompts for the specification, it is very likely that the implementation is correct. Other tools, such as static analysis tools, could also be included in the second loop, to ensure that the code has certain properties. Tools to analyze code quality could be used to select the best implementation among a certain number of candidates produced in the code production tool, too.

The name of the approach, Specification-Test-Check Loop (STCL) comes from the phases of the improvement loop described above.

4.3.2 Illustrative example

An more detailed example of how a process based in the the STCL approach could work is as follows:

(a) Users specify a program in natural language.

(b) The system automatically generates a set of tests.

(c) The system augments the tests by using mutation testing.

(d) The system produces a natural language explanation of the tests.

(e) Users adapt and adjust these tests (as "use cases" in natural language).

(f) Once satisfied with the tests, the system generates code, tests it, and run some analysis tools on it. If it works, and the analysis tools give good results, the process concludes.

(g) If it doesn't work, or the analysis tools show areas for improvement, the system proposes how to "fix" the code to pass the tests or to improve the results of the analysis, using this as input for re-executing step (e).

The goal is to converge on a final code that meets the specified requirements, passes all tests, and results in a positive analysis by the automated tools.

It is important to notice how, in this example, all the interaction with the system is via natural language, and users do not need to see the actual source code being generated: it is enough for them to check the explanations that the system produces.

For running these kind of loops, it is likely that standard generative models for code are not enough. Likely, we will need fine-tuned versions of foundational models, trained with execution logs of tests, with results of analysis tools, and with changes of the source code to pass tests or improve the results of the analysis. Some preliminary explorations show that fine-tuning these models can achieve reasonable results that enable the approach.

4.4 Common toolkit and specific tools

Many of the activities to be performed for supporting both approaches are similar, or in some cases, exactly the same. Therefore, it makes sense to identify modules useful for both the SBM and the STCL approaches, so that both of them can be supported with less coding effort.

Therefore, it will be important to design the whole system with this reusability in mind. In fact, it will be useful for the adoption of the system to design it in a way that can be also used for other approaches, which other research teams may want to try. So, we will extend this idea of building reusable components, which can work together in different ways, as much as possible.

Although an exhaustive exploration will be needed of which modules can be designed and built in a way that are useful for both approaches, we have already identified some important modules and characteristics of them:

- Composable execution of generative models, in a way that several inputs coming from other modules can be used as parts of the prompt, with the outputs being fed also to other modules that will perform certain actions with them (for example, running tests of a piece of code, or executing an scenario). Currently, we envision an extension of the *langchain* architecture[1], for example.
- Modules for running non AI-related subsystems, such as testing or program execution, but in a way that their outputs can be part of a prompt, and their inputs can be produced by generative models.
- Generative AI models for producing user interfaces, which will be useful both for the general interaction with humans, but also for the interactive design of scenarios used in SBM.
- Modules for facilitating the implementation of empirical tests, which will make it much easier to evaluate and validate results.

4.5 Case study

For evaluating and validating the final results of the project, we will run a case study, as much realistic as possible, so that we can test both the SBM and the STCL approaches in a practical environment. This will allow us not only to check how useful and sound they are, but also to compare how both could be more or less adapted to specific use cases.

4.5.1 Overview of the case study

Our case study focuses on the development of a complex enterprise software application that leverages a microservices architecture [26] and Event-Driven Architecture (EDA) [23]. This type of architecture is characterized by its scalability, flexibility, and responsiveness, making it suitable for modern enterprise requirements. However, this architecture is chosen not to test

[1] `https://www.langchain.com/`

its scalability or flexibility per se, but as a typical example of the complex software systems OSOCO routinely develops in its industrial projects.

The case study will consider a real-world application: an Application Tracking System (ATS) named *Contestia* [28], developed by OSOCO. This system exemplifies a complex enterprise software application leveraging a microservices architecture and Event-Driven Architecture (EDA).

Contestia is designed to manage various complex business processes and workflows, utilizing multiple independent microservices that communicate through event-driven mechanisms. Instead of an hypothetical example, employing Contestia offers a tangible, practical scenario that reflects the real-world complexities our tools aim to address.

For this case study, we will not reimplement the entire Contestia system due to its extensive scale. Instead, we will concentrate on the central service of Contestia (*contestia-core*). The ultimate test will involve replacing the current implementation of this core service in a test environment with one generated using our SBM and STCL tools. This approach will allow us to directly assess the practicality, efficiency, and accuracy of the tools in a real-world application.

By focusing on a specific, functional component of Contestia, we can effectively demonstrate the capabilities of our tools in enhancing and streamlining the development process, particularly in complex systems typical of modern enterprise environments. The case study will provide valuable insights into how our tools can be integrated into existing software development workflows and their impact on improving system design and implementation.

4.5.2 Goals and outcomes

The case study's main goal is to demonstrate the efficacy of SBM and STCL Tools. We will use the developed tools to create, modify, and maintain various components of the application, showcasing how these tools improve efficiency, accuracy, and collaboration in the software development process.

- **Validate the Utility of SBM and STCL Tools**: Demonstrate how the developed tools facilitate the creation, modification, and maintenance of different components of the application, specifically highlighting their impact on enhancing efficiency and collaboration in software development.
- **Assess Tool Viability for Future Commercial Product Development**: Evaluate the potential of these tools for integration into OSOCOs product development pipeline, aiming to create a future commercial product targeting the CASE sector.

The expected outcomes for the case study will be:

- **Successful Development of an Enterprise Application**: Completion of a fully functional enterprise application that meets specified business requirements and demonstrates the practical application of the tools developed in the project.
- **Insights into Tool Effectiveness for Industrial Application**: Gather insights into the tools' effectiveness in a real-world software development setting, identifying areas for improvement and potential for commercial product development.
- **Documentation and Commercial Viability Analysis**: Document the development process comprehensively, analyzing the case study for its potential to inform the development of a commercial CASE product.

4.6 Key aspects of the proposal

From the scientific point of view, the key aspects of the proposal, which advance the state of the art in several areas, are:

- Amplification of domain-driven design practices, by using generative AI to extend the the domain discovery process into a more dynamic and iterative modeling practice.

- Extension of scenario-based modeling, by using generative AI to automatically build models from user-designed scenarios.
- Extension of test-driven development, by using AI generated tests to assist in the creation and modification of software.
- In general, improving the code produced by generative AI, making it more reliable, and more in line with the requirements of the developer.

From the technological point of view, the key aspects of the proposal, which will produce innovative outcomes of practical use are:

- Reuse of the current research in generative AI models, and generation of code using LLMs.
- Practical development processes for using AI with low-code or zero-code approaches, enabling domain experts to develop and maintain software.
- Set of tools, tested in real-live used cases, implementing those processes, and ready to be used in at least some application domain.

All of this will allow the industrial partner to improve its competitiveness in the production of software, and to open a new business line, providing services to customers willing to use the produced tools for their own businesses. This new line will be reinforced by the availability of the produced tools as free, open source software, which will raise the need by some of their users of services related to their efficient use, and maintenance, which will be provided by the industrial partner.

5. Methods, work plan and budget

5.1 Methods and strategies

This project is organized following two kinds of strategies:

- "solution-seeking" research and technology strategy[2] for the research and technical objectives (O-Research-SBM, O-Research-STCL, O-Tech-Toolkit, O-Tool-SBM, O-Tool-STCL), which will be addressed as "practical problems" [37]
- "knowledge-seeking" strategy[3] for the validation and evaluation of the results of those strategies: the systems produced to achieve the objectives for both the SBM and STCL approaches, and the overall produced sytem and processes in some specific scenarios (O-Validation).

For each of the research and technical objectives, we will design an activity, which will implement the strategy to reach it. For each of the SBM and STCL approaches, we will also design an activity for evaluating it. For validating the final output of the project (O-Evaluation) we will design another activity. The following table summarizes this organization:

Objectives	Activities	
	Solution-seeking	Knowledge-seeking
O-Research-SBM O-Tech-SBM	A-Research-SBM A-Tech-SBM	A-Validation-SBM
O-Research-STCL O-Tech-STCL	A-Research-STCL A-Tech-STCL	A-Validation-STCL
O-Tech-Toolkit O-Validation	A-Tech-Toolkit	A-Validation

Solution-seeking activities will therefore produce the description of the processes to support SBM and STCL, the tools to implement them, and the common toolkit used to facilitate both implementations. Each of the SBM and STCL approaches will be empirically evaluated to ensure their feasibility and soundness. Together all these activities they will produce a system suitable for both SBM and STCL, which will be integrated by A-Tech-Toolkit into the final proof-of-concept system usable for the case studies used for validation. Figure 5.4 shows these activities and their relationships.

In addition, we will have feedback loops from the knowledge-seeking acitivities to the solution-seeking activities, to incrementally improve their outputs. This distinction between research and technical activities, on the one hand, and between solution-seeking and evaluation (knowledge-seeking) activities, is expected to improve empirical validation, by having a clear separation of aims and responsibilities, but at the same time, letting experience with the results help to improve the results of the project.

From a temporal point of view, the project will start with research activities, with technical activities starting as soon as research results allow them. Once technical activities start to produce prototypes, validation activities of SBM and STCL approaches will start. Since that moment on, all three kinds of activities will go on in parallel, ensuring the needed feedback for incremental improvement. At some point, research activities finish, hopefully with sound processes as an output. Later, technical activities finish too, letting the last period of the project for ensuring a proper validation with the use cases (even when for some time, validation, technical and research activities will also run in parallel, to ensure that some information obtained from the use cases can be retrofitted).

The overall methodology for each of the SBM and the STCL approaches (with their corresponding research, technical and validation activities) will be:

[2] "To design or develop new or improve existing solutions that can help to overcome or ameliorate challenges, bottlenecks, and other problems in the development of software systems and supporting processes" [32].

[3] "To generate or propose scientific claims and to evaluate and validate those claims. This may also include the development of instruments [...] with the specific purpose of supporting or enabling these knowledge-seeking activities" [32].

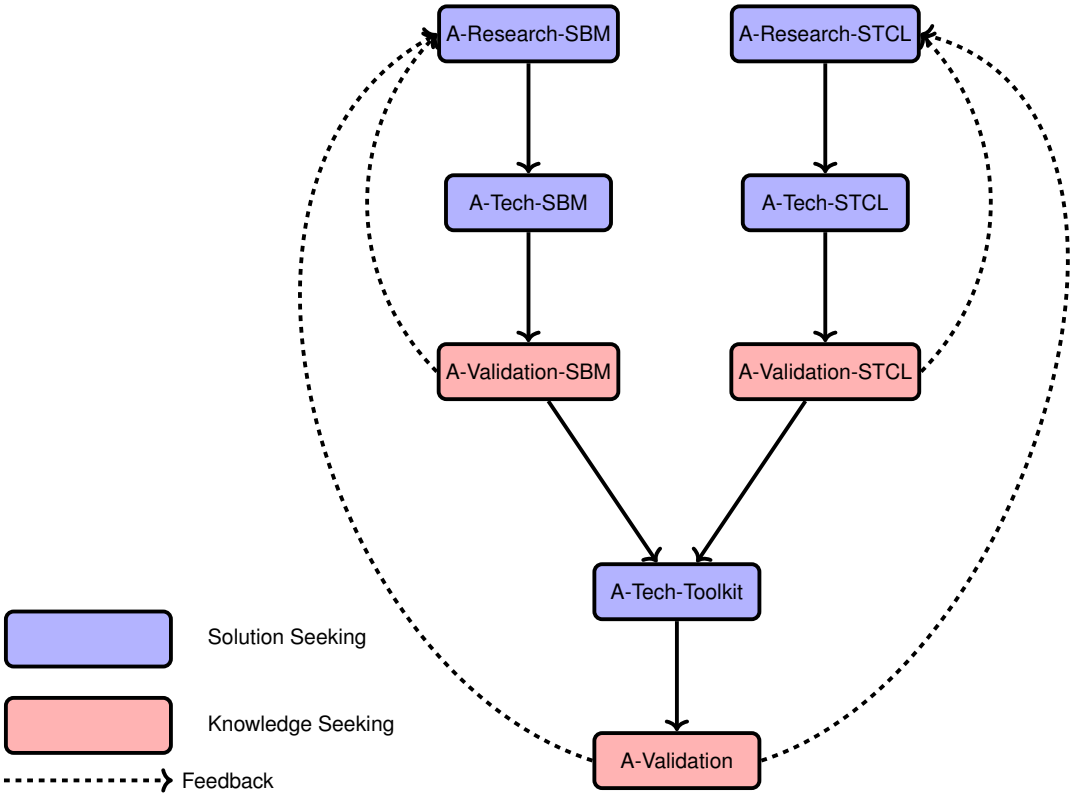


Figure 5.4: Diagram of the project activities.

- 1. Exhaustive analysis of the problems to solve, and of the possible solutions for them. This analysis will be based on our expertise in the field, complemented with a focused analysis of the published state of the art, and specially the most recent publications, following the Systematic Literature Review method [20].
- 2. Select the most promising solutions, refine them, and build a tool implementing them with the detail needed for an empirical evaluation, identifying common modules for reusing them as a part of the toolkit. Since this step will be recurrent, and to some extent incremental, software developed in it will be modeled as an agile sprint (as we have done extensively in previous projects).
- 3. Validate the solution implemented in the corresponding evaluation activity.
- 4. Analyze results of the empirical evaluation, adding them to the current analysis of the problem, and repeat from step (2), until validation is satisfactory in the scenarios considered.

Step (1) will be done in the corresponding research activity, step (2) in the corresponding technical activity, with the support of A-Tech-Toolkit (all of them solution-seeking activities), and steps (3) and (4) in the corresponding knowledge-seeking activities. The details of the analysis, selection of solutions, and validation will be provided later, in the description of the work plan.

The methods used for validation will depend on the nature of the activity (settings mentioned below follow the definitions found in [32]):

- *A-Validation-SBM*
  - We will conduct a qualitative experiment designed to gather in-depth user feedback and insights into the tool's usability, effectiveness, and overall user experience. This experiment will involve a diverse group of users —including software developers, and individuals with minimal programming experience— who will interact with the tool in a series of tasks, followed by interviews and surveys. The key questions addressed will be: (i) Comprehensibility: How understandable is the play-in process? Are users able to easily grasp how to specify system behavior using the tool?; (ii)

Natural interaction with AI: Does the integration of generative AI in the multimodal UI feel natural? How do users perceive the AIs role in aiding the design process?; and (iii) Adaptability to changing requirements: How easy is it for users to evolve the software's functionality when new requirements arise? The feedback from these sessions, focusing on these key areas, will be crucial in assessing the tool's practicality and identifying areas for enhancement.

  - We will design a quantitative experiment to compare the effectiveness of our Smart Play-In tool, enhanced with generative AI, against the conventional Play-In process in SBM. This experiment aims to provide a clear, data-driven understanding of the improvements our tool offers in efficiency and modeling accuracy. The experiment will involve measuring the time participants take to complete specific modeling tasks using both methods, as well as the number and accuracy of scenarios they can specify. Participants, including both experienced and novice modelers, will undertake a series of tasks designed to mimic common challenges in software development. Their performance with both the conventional Play-In process and our Smart Play-In tool will be recorded and analyzed.

- *A-Validation-STCL* will use both a Laboratory Experiment setting, consisting of a large number of "code production experiments", and a Experimental Simulation setting, consisting of a much smaller "code production simulations".

  - For each of the code production experiments, a specification and a testsuite will be defined. The experiment will consist in running the STCL automatic process, asking the ML model to produce code according to specifications and the testsuite, running the tests on that code, feeding back the log of running the tests to the ML model to produce a new version of the code, and so on. The analysis of the experiment will check if the number of tests passed increases, and if the process eventually congerges to all tests passing. This experiemwnts will allow not only for validating the idea, checking if this automatic process can improve the code, but also for improving the prompt templates, trying several fine-tunings, and in general for refining the process. Datasets such as TACO [21] will be used to ensure that a large quantity of good-quality experiments can be run.
  - For each of the code production simulations, a specification will be defined. Based on it, a simulation of the complete STCL process will be performed, with humans in the loop for controlling the output. The results will be evaluated in terms of accuracy of the results, quality metrics obtained from the produced code, qualitative evaluation by the human involved, and comparison between different runs. Both specifications from datasets and tailor-made specifications will be used, to explore different application domains and kinds of implementations. This evaluation should help to better understand to which extent the process is practical, if it produces reasonable code for the original specification, and how much usable the whole process is.

- *A-Validation* will use a Field Experiment setting, consisting of real use cases. The industrial partner will run use cases, with domain experts leading the experiments, producing prototype systems for those cases using the tools developed by the project. Therefore, both approaches (SBM and STCL) will be evaluated, by using the integrated system, including the common toolkit with the specific SBM and STCL modules, and studying the qualitative and quantitative characteristics of the resulting system and of the process to produce it. This method will therefore evaluate the final results of the project.

Some of these validation studies will be performed as Registered Reports [4] to minimize bias in the perception of successful validation by researchers, and to ensure publication of negative results, if any. In any case, the usual practices of open science, including publication of intermediate results, and publication of detailed reproduction packages, will be followed, to ensure transparency and reproducibility of results.

5.2 Activities

In this section, the activities of the project are described. In addition to the technical activities already introduced, we will have an additional non-technical activity, *A-Management*, for dealing with all the management issues, including the kick-off of the projects and the acquisition of equipment, tracking of progress, risk management, overall evaluation, personnel management, open science practices and specific dissemination activities. In the context of this Activity we will have monthly meetings of the combined research team (including both partners).

5.2.1 A-Research-SBM

Description: To design a process for achieving *O-Research-SBM*.

This activity delves into integrating generative AI with Scenario-Based Modeling (SBM), with a special focus on researching multi-modal user interfaces. The aim is to develop innovative methodologies, algorithms, and theoretical frameworks that not only enable AI to assist in SBM but also facilitate user interaction through diverse modalities, such as visual elements, natural language processing, and intuitive graphical interfaces.

5.2.2 A-Tech-SBM

Description: To design and implement the tools for achieving *O-Tech-SBM*.

In this activity, the insights and methods derived from the research phase are transformed into practical technological solutions. The primary output is a user-friendly tool that incorporates generative AI to facilitate SBM. This tool aims to simplify the creation of models for microservices-based reactive applications, making scenario-based modeling more accessible to software engineers and domain experts.

5.2.3 A-Validation-SBM

Description: To evaluate both the process and tools designed and built in *A-Research-SBM* and *A-Tech-SBM*.

This activity involves the empirical qualitative and quantitative evaluation of the tool developed in the A-Tech-SBM phase. It includes user testing, performance analysis, and usability studies to assess how effectively the tool meets the needs of its users and the goals of SBM. Feedback loops from this validation phase will inform further research and technology development, ensuring continuous improvement of the tool.

5.2.4 A-Research-STCL

Description: To design a process for achieving *O-Research-STCL*.

This activity is devoted to design the development and maintenance processes based on the Specify-Test-Check loop (STCL) approach. The main aim is to develop an innovative theoretical frameworks, and empirically validate it, to produce code and code changes following instructions from human developers, but at the same time ensuring the soundness of the resulting code, and its adherence to the specifications of the humans. The process is also expected to help humans to realize constraints and invariants to have into account that they had not realized previously, with the help of AI agents.

5.2.5 A-Tech-STCL

Description: To design and implement the tools for achieving *O-Tech-STCL*.

This activity will focus on the production of a proof-of-concept tooling for testing the STCL approach. It will produce both the user interface and the different modules that need to be chained (prompt-generation, ML model execution, test generation and running, retrofitting of test-run logs, automated tools to check for constraints or to rank different versions of produced code, etc.) to implement this approach. This activity will also be in charge of the fine-tuning and other techniques for improving the performance of the ML models used. The final result will be a system that can interact with a human, who will produce specifications, supervise the production of tests, and react to suggestions of new tests or changes to the specification, until the produced code is the code intended.

5.2.6 A-Validation-STCL

Description: To evaluate both the process and tools designed and built in *A-Research-STCL* and *A-Tech-STCL*.

This activity will evaluate and validate the process defined in *A-Research-STCL*, via the execution of components (or the whole system) produced by *A-Tech-STCL*. The empirical experiments used will be of two kinds: fully automated experiments, to evaluate under which conditions the production of code converges towards passing all the tests, and qualitative experiments conduced with human subjects, to evaluate how they behave with the system, and to which extent the system produces the code they intend to produce.

5.2.7 A-Tech-Toolkit

Description: To design and implement modules common for supporting *A-Tech-SBM* and *A-Tech-STCL*, and integrating all of them into the complete prototype.

This activity will identify common modules useful both for the SBM and for the STCL approaches, and a general architecture for supporting processes in which humans and AI agents collaborate to produce code and code changes. The overall system produced by this activity will be designed to minimize the code produced by *A-Tech-SBM* and *A-Tech-STCL*, which will only need to devote to the aspects specific of their approaches. The resulting system will also be designed to easily support other different processes, so it can be a platform of interest for other research groups, willing to check other alternative methods. The toolkit will also feature components specifically designed to facilitate the implementation of empirical tests, such as log collecting and analyzing modules, user interface modules oriented to collect data from humans participating as subjects in evaluation experiments, and modules to automate the execution of large quantities of similar runs when constraints and variants of the approaches are explored.

5.2.8 A-Validation

Description: To evaluate the overall results of the project, and in particular the feasibility, advantages and potential problems of building software systems with the SBM and the STCL processes.

This activity is crucial for empirically assessing the effectiveness of the developed tools in the SBM and STCL lines of our project. The validation will be conducted through a case study focusing on Contestia, a sophisticated Application Tracking System (ATS) developed by OSOCO. Contestia exemplifies an enterprise software application built upon a microservices and Event-Driven Architecture (EDA), reflecting the complexity and demands of modern software systems.

5.2.9 A-Management

Description: To manage the project, including all management, coordination, and procurement actions, and to design and execute the detailed dissemination plan, the open science plan, the technology-transfer plan, and the intellectual property plan.

5.3 Work plan and Schedule

As we introduced in Section 5.2, the project is structured in nine activities. It will last for 36 months, organized in three feedback-loop stages, and a final evaluation and exploitation stage:

Stage	Start–end (months)	Description
Inception	1–12	Early version of process and implementation
Refinement	10–21	Refinement of process and implementation
Completion	19–30	Final version of process and implementation
Evaluation	28–36	Overall evaluation (case studies) & exploitation

Each of the first three stages will run research, technical and validation activities partially in parallel, but with different intensities as time progresses. During the first four months, effort will be mostly devoted to the research activities (*A-Research-SBM* and *A-Research-STCL*), from month 5 to 9 of the stage, to the technical activities (*A-Tech-SBM* and *A-Tech-STCL*), and finally, the last three months will be mainly devoted to the evaluation activities (*A-Validation-SBM* and

A-Validation-STCL). However, all activities will remain with some level of activity during the whole stage, to allow for quick consideration of feedback, and reaction to unexpected problems. There is also some level of overlapping of stages because we expect that a certain stage can start while the previous one is still being evaluated.

During the *Refinement* stage *A-Tech-Toolkit* will be activated, to identify and build common components, and to integrate all software components in complete prototypes (in the *Inception* stage this will not be needed, since prototypes will still be preliminary and separate for SBM and STCL). It will remain active until the end of the project, although with low intensity after the completion phase, to ensure response to needs of the final evaluation phase.

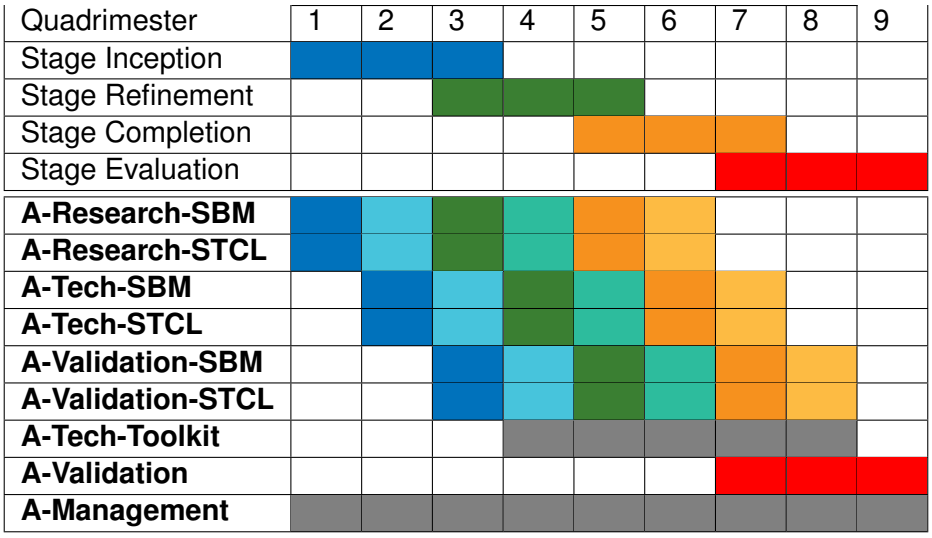
A-Validation will be active during the last 9 months of the project, to perform the final evaluation of the project results. *A-Management* will be active during all the project, with a special emphasis on exploitation plans and final dissemination during the final stage.

Feedback loops will therefore be synchronized for all research, technical and evaluation activities. For organizing the progress of the project, we will define milestones at the end of each of the stages, where outputs will be collected to track the overall progress of the project.

The schema with the duration of Activities, and the participation of each partner in it, is as follows (in boldface, the partner responsible for each activity).

Activity	Duration (months)	Start–End (months)	OSOCO (effort)	URJC (effort)
A-Research-SBM	24	1–23	**60%**	40%
A-Research-STCL	24	1–23	20%	**80%**
A-Tech-SBM	24	5–27	**80%**	20%
A-Tech-STCL	24	5–27	30%	**70%**
A-Validation-SBM	24	9–31	20%	**80%**
A-Validation-STCL	24	9–31	**30%**	70%
A-Tech-Toolkit	21	13–33	**50%**	50%
A-Validation	10	27–36	**70%**	30%
A-Management	36	1–36	**60%**	40%

The next Gantt chart shows the scheduling of activities over time. In it, we have represented approximately (by quadrimesters) the most active activities in the color corresponding to the stage, and in light colors the periods of less activity. In gray, tasks not directly related to stages. The selection of colors try to make it more clear the shift of effort as stages progress, not making the chart too complex.



Deliverables to be produced are of several types: **Scientific reports (SR)** (including theoretical and methodological details), **Tools (T)** (complete toolsets, and specific set of tools used for the experiments), **Data sets (DS)** (complete datasets produced in experiments), **Research papers (RP)** (linked to results of the two last stages), and **Management reports (MR)** (including an updated risk assessment and plan for correcting deviations).

Milestone	Date (month)	Name	Type
M1	12	SBM process, tools and validation	SR, T, DS
M1	12	STCL process, tools and validation	SR, T, DS
M2	20	SBM process, tools and validation	RP, T, DS
M2	20	STCL process, tools and validation	RP, T, DS
M2	20	Mid-term management report	MR
M3	28	SBM process, tools and validation	RP, T, DS
M3	28	STCL process, tools and validation	RP, T, DS
M4	36	Final system, use cases	RP, T, DS
M2	36	Final management report	MR

Research papers are produced once SBM and STCL processes and tools reach a reasonable maturity, and are evaluated (after the two last stages), and at the end of the project, presenting the overall approach, and the results of the use cases. At the end of the project, the main output will be the whole integrated system, useful as a prototype for producing software systems close to ready for production.

5.4 Critical aspects and contingency plan

The project will implement a continuous risk monitoring procedure, with periodic (every six months) progress reviews and comparison with plans, by all researchers and developers collaborating with the project.

Risks for the project can be divided in internal (due to matters of the project), and external. The internal risks that we have identified (with their corresponding contingency plans) are:

- Failure to design a SBM or STCL process which fulfills requirements, and specifically, which improves the code directly produced by the generative model in a single round, and lets the user stay in control guiding the details of the functionality of the produced code. (Likeness: High). This is likely the highest risk, linked to the innovative nature of the project. *Contingency plan:* If the problem arises, it will be during one of the stages: still we can try during the remaining stages to design a proper process. In any case, one of the reasons for trying two approaches is to at least have one successful approach if the other fails. Therefore, if we see no way of producing the right process for one of the approaches, we would focus on the other.
- Failure to produce the right tools, that let users of the system work according to the SBM or STCL processes in a practical and useful way. (Likeness: Medium). Given the experience of the industrial partner in building software, it is unlikely that this happens from a functional point of view, but we could have performance problems, if the resources needed to produce code by generative models are too much expensive in terms of execution time, hardware, etc. *Contingency plan:* Explore other fine-tuning or training methods, which are more efficient, or redesign the process to be more efficient.
- Failure to produce the integrated system, in a way which is useful and shows the whole capabilities of the approaches. (Likeness: Low). This could happen if the integration does not work, or if the usability of the whole system is not good enough. *Contingency plan:* Being careful in the monitoring of the integration and usability problems, to prevent the problem, and devote more resources to integration, so that it can be done more efficiently.
- Experiments are more difficult to run than expected (Likeness: Low). The project is heavily based on a search of empirical results, and thus experiments are fundamental. We hope that they are easy to design, implement and run, so that we can run many of them. *Contingency plan:* To devote a significant effort to ensure that support and automation for experiments is good enough. We also have different kinds of experiments, in the hope that at least some of them can be run.

The main external risks we have identified are:

- Not enough progress in the availability of new generative models with increasingly better capabilities for generating code (Likeness: Low). Since the project is not going to produce

its own generative models, but will rely on those available publicly, it may happen that those are not of good quality, good enough for the purposes of the project. *Contingency plan:* To test as much generative models as possible, with the hope of finding the one which is adequate for our needs. Also, to invest more in fine-tuning and other techniques which can improve the capabilities of the model.

- Problems recruiting personnel (Likeness: Low). Given the time and economic constraints, maybe the academic partner cannot recruit personnel with the adequate training and expertise. *Contingency plan:* Since delaying recruitment until the right candidates are found is not an option, because we need to run the project according to the plan, in the specified time frame, we will mitigate this risk, if it materializes, by providing adequate training to the recruited personnel, when that is within the capacities of our team.
- Problems finding subjects for the use cases (Likeness: Medium). Finding subjects for the final use cases (domain experts in areas of expertise of the industrial partner) is fundamental for the execution and evaluation of the use cases. *Contingency plan:* The industrial partner will prepare in advance the use cases, planning as much as possible the availability of the needed domain experts.
- Personnel leaving the working team (Likeness: Low). People hired by the academic partner do not have a permanent relationship with the University, and could leave during the execution of the project. *Contingency plan:* Most of the critical tasks will be assigned to permanent personnel, and quick hiring processes will be prepared to recruit more people in case some leave the project.

7. Impact

7.1 Expected direct impact

This project, rooted in the integration of generative AI with software engineering practices, is expected to make significant contributions across multiple domains. Its impact is envisioned to span scientific advancement, technological innovation, social development, economic growth, and market potential.

This impact will be both national and international. From a national point of view, the know-how will be developed locally, and thanks to the participation of OSOCO, it will have a clear path to the production sector, helping to increase both the scientific and industrial competitiveness of national actors. At the international level, the participation of the URJC team will ensure a worldwide impact, seeing at the same time our current positions strengthened, being reinforced as a center of know-how (scientific or industrial) in this field.

7.1.1 Tool adoption in OSOCO's engineering projects

In the short term, the tools developed in the project will be integrated into OSOCO's existing software engineering practices. This integration will enhance the efficiency and efficacy of the company's consultancy and enterprise software development projects. The adoption of these tools by OSOCO's engineers will serve as a real-world testbed, providing immediate feedback for refinement and demonstrating practical applicability.

7.1.2 Widespread adoption and open dissemination

As the project results and tool codes will be openly disseminated and available as Free and Open Source Software (FOSS), we anticipate a gradual but steady adoption of this novel software development approach. This approach, characterized by reduced code prominence and open to collaboration from all stakeholders, is expected to gain traction among practitioners of Domain-Driven Design (DDD) and domain exploration techniques such as Event Storming, Example Mapping, Domain Storytelling, and CoMo (Collaborative Modeling).

The open-source nature of the tools will encourage experimentation, adaptation, and adoption, fueling innovation and knowledge sharing in the software development community.

7.1.3 Commercial product development and advanced design environments

Looking further into the future, OSOCO plans to leverage the insights and technologies developed in the ADVISE project to create a commercial product. This product is envisioned as a sophisticated Software Design Environment equipped with Augmented Reality (AR) and spatial computing capabilities. Such an environment will not only transform the landscape of software design tools but also open up new possibilities for interactive, immersive, and collaborative software development.

7.2 General technical impact

Beyond the direct application in software engineering, we think the project transcends the boundaries of traditional software engineering, promising significant societal impacts in the realms of *end-user programming*, *participatory design*, *design thinking*, and *systems thinking*.

End-User Programming and Participatory Design By enabling non-technical stakeholders to actively participate in the software development process, the ADVISE project aligns with concepts like end-user programming [25] and participatory design [8]. This approach fosters inclusivity, allowing users to contribute to software creation and customization, thus bridging the gap between developers and end-users. This democratization can lead to software that better reflects the needs and perspectives of a broader user base.

Design and Systems Thinking In educational settings, the tools and methodologies developed in the project can be instrumental in teaching design thinking and systems thinking. As noted by Terry Winograd [38], there's a growing need to integrate these concepts into software development education. The project's focus on intuitive and collaborative design approaches aligns well with these educational goals, preparing future software engineers to think holistically and creatively.

AI in Software Engineering Scientifically, the project advances the understanding of how AI can be effectively integrated into complex problem-solving and creative processes, setting a precedent for future research in AI-assisted software engineering.

7.3 General scientific impact

This project will advance the state of the art a very important area for the future of softwar engineering: the collaboration of humans and AI agents in the construction of large, complex software systems. We expect that the approach used by the project, directly rooted on empirical experiements to ensure the soundness of results, and the industrial collaboration to ensure their relevance and practical use, causes a great impact in the field.

More specifically, we will research two approaches (SBM and STCL) to a very important and current problem: how to design processes that, keeping the humans in control, make the use of AI agents as much productive as possible in the production and maintenance of software systems. We expect that both of them allow for developers and domain experts to better interesact with those agents, in a way that the resulting product satisfies their requirements, being produced in an almost completely automatd way. This goes several steps beyond raw production of code using generative models, being in the same domain of software developing methodologies for teams of human develpers (in which our approaches are inspired).

We also expect that the software system implemnting our two approaches, beng distributed as FOSS, is used by other resarch teams to test and improve our results, leading to a de-facto standard platform which can be used to test different approaches of using generative AI for the production and maintenance of software. In fact, our “common toolkit” is designed to be a conveyor for this strategy: with little effort, other research teams can use it to design their own processes inclding AI-based code generators.

7.4 General social and economic impact

The project is expected to have a substantial economic impact, since addresses a relevant problem in one of the most interesting areas for the software industry: efficient production and maintenance of software systems. The participation of a national company with a clear interest in its international expansion will help to convert this potential in increased competitiveness, helping them to participate in the international market of suppliers of technology to software producers, specially in the secondary sector (industries using software to provide their main services), and in particular for tourism, automotive, aerospace or health, where the importance of the production of software is clear, and the difficulties for domain experts to efficiently participate in that process are a fundamental problem with a great impact on their competitiveness.

From the social point of view, the technology produced by the project will help to involve domain experts, not fluent in software development, in the production and maintenance of software. We expect that it will facilitate a better understanding of the complexities of software to people with other backgrounds, thus helping to the better use of technology, and for a better understanding between IT professionals and other professionals.

The application of the technology to real deployments will also help in the detection of faulty, or insecure, component versions, and in the production of trustable systems. This will help to have more secure and robust deployed applications and services.

The fact that all the technology produced by the project will be FOSS, and actively maintained with the help of a commercial company, will also help in its transfer to the productive sector, in a pattern that the research team has already explored, with success, in the past.

7.5 Transfer of results

This project is designed to favor technology transfer in two complementary ways:

- Transfer to the research community. From a scientific point of view, the resulting description of processes, tools and data sets will be publicly available as open access documents, FOSS and open data, fostering that other researchers reuse them, and reproduce our results. We expect that the software and datasets produced will become in the medium term, widely used in the academic community.

- Transfer to the industry. From an industrial point of view, OSOCO, the company collaborating in this project, will include in their portfolio the results they find interesting from a commercial point of view. This will ensure a commercial offer and the adequate support for the technology transfer to the productive sector.

It is worth mentioning that the consortium has agreed to release all software produced as a part of this project as free, open source software (FOSS). Therefore, there is also a transfer path through the transfer channels enabled by FOSS. Since the project intends to produce a complete system, it is expected that this system will be used as a basis for improvements and complements. We expect that this will lead to the origin of a new concept of Integrated Development Environment, with generative models at its core, using the procedures designed in this project. OSOCO plans to harness this evolution to exploit business opportunities in the maintenance, customization and evolution of the system, thus creating a powerful feedback loop with its exploitation strategy,

The consortium also adheres to the principles of open science, including all the results of the project, also those that could be considered of commercial interest. In particular:

- All the publications, including academic publications in research journals or conference proceedings, gray literature produced, and all technical reports describing the processes, the experiments and their results, will be published as open access documents in the institutional open archive of URJC, BURJC digital, and in one thematic repository, Zenodo.
- All datasets, including those used in the experiments and in the use cases, and also those used for training and fine-tuning, will be datasets granting permission for those tasks, and when produced by the consortium in the context of this project, will be released as open data in the Open Data Archive of URJC, eCiencia, and in Zenodo.
- All the software, fine-tuned models, and tools produced by the project will be released as FOSS (free, open source software), in our own repository in GitLab.
- Documents intended to be understandable by the public in general will be produced, seeking that the society at large can understand the aims and results of the project.
- Reuse of all our materials, including reproduction of our experiments, and use for any kind of tasks of the software produced, will be enabled by using FOSS licenses, and encouraged providing a reasonable support to third party users.

For all these matters, we will use the support of the Office for Open Knowledge of the URJC.

The consortium considers inclusivity and ethical practices as an important aspect of the project. In addition to other more generic measures, the project will devote special attention to:

- Being as much inclusive as possible in all steps of our research and software production cycles. This will include analyzing and mitigating possible bias in training and fine-tuning datasets, and checking the outputs of ML models for inclusivity-related bias with respect to gender, age, ethnic origin or nationality, etc.
- Mitigate bias en experiments including human subjects, by ensuring a reasonable diversity of them with respect to gender, age, ethnic origin, etc.
- Consider as much as possible languages other than English. In particular, some of the tests will be done with other natural languages, and specifically, Spanish.
- Follow the highest ethical standards in our research and development. In particular, any research involving human subjects will be submitted for approval to the Ethics Committee of the URJC.
- Follow inclusivity measures for hiring and human resources matters, ensuring that minorities and underrepresented genders and communities have opportunities to participate in the project in good standing.

For all these matters, we will use the support of the Diversity Unit and the Equality Unit of the URJC.

7.6 Dissemination plan

The dissemination plan is tightly linked to the technology transfer strategy shown above, using these means: **traditional Internet-based dissemination means**: web site, social network channels, blog posts, etc.; participation in seasonal **Practical Schools** for junior researchers; **dissemination to the software development communities** which could have interest in our results, and in particular in the communities producing ML models, and in those interested in the creation of automatic tools to support software development, in their own conferences and venues; collaboration with other companies in the **definition of commercial services** based on the results; and **publication of datasets** in appropriate venues, so that other academics can reuse and improve them.

In addition, traditional **academic dissemination** in journals and conferences will be performed for the most relevant scientific results. Targeted journals are IEEE Trans. on Software Engineering, Empirical Software Engineering, Information & Software Technology, and Journal of Systems and Software. Targeted conferences are ICSE, FSE, MSR and ICSME. According to the workplan, papers for conferences and journals will be produced for each stage, and when the final results of the project are produced. We also intend to propose specific workshops and hackathons on the matter, and the contribution of datasets and tools to these academic venues.

This way, we will convert both the tools and the datasets into conveyors for the transfer of results, both to the academic community and to the industry, supported by a commercial strategy. It is also important to notice that our team already has experience with all these strategies, that were used, for example, to popularize our GrimoireLab toolset.

We will also leverage the contacts of OSOCO and URJC in different communities of practice and national and international associations and organizations that will help to disseminate results at the national and international level. For example, OSOCO coordinates and participates, among others, in DDD Hispano[4], Madrid Smalltalk User Group[5] and Collaborative Modeling (CoMo) Madrid[6], and the URJC is involved in the esLibre Conference[7]. At the international level, OSOCO is industrial member of the Pharo Consortium[8], and is a regular participant at ESUG[9]. The URJC team participates in the CHAOS[10], and has good relationship with the Linux Foundation[11], the Eclipse Foundation[12] and the Apache Foundation[13]. We intend to explore the organization of joint events with all these groups, and participate in their conferences and expos.

7.7 International dimension

In addition to the groups and contacts mentioned above, the URJC team contributes with a strong network of international collaborators, both in academia and industry. The team will exploit this network to foster collaboration based on the processes defined in this project, and the software developed in it. We expect that many other researchers will be interested in running joint experiments in our platform, which will help to have joint publications, which in turn will ensure that the project, and its main results, will be known, first hand, by several world-class research groups.

Since the team also has good relationship with organizations such as the Linux Foundation, the Eclipse Foundation and the Apache Foundation, which are in themselves technology hubs, and meeting points for the industry in many domains, the exposure of our results to the international technological landscape is granted. In particular, the team is already working with the Eclipse Foundation in preparing proposals at the European level which would be complementary of this project, and in presentations in their conferences and working groups about the

[4] https://www.meetup.com/dddhispano/
[5] https://www.meetup.com/MadridSUG/
[6] https://www.meetup.com/collaborative-modeling-madrid/
[7] https://eslib.re
[8] https://consortium.pharo.org/
[9] https://esug.github.io/
[10] https://chaoss.community
[11] https://www.linuxfoundation.org/
[12] https://eclipse.org
[13] https://apache.org

approaches described in this project.

In the long term, we envision that the technology we will develop, being distributed as FOSS, will find a home in one of these foundations (likely the Eclipse Foundation, which has a strong interest in tools for supporting software development and maintenance), which will help to foster a worldwide community of users and, with time, developers interested in developing the technology further.

Bibliography

[1] Malak Abdullah, Alia Madain, and Yaser Jararweh. Chatgpt: Fundamentals, applications and social impacts. In *2022 Ninth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 1–8. IEEE, 2022.

[2] Alberto Brandolini. Introducing event storming, 2013.

[3] Yihan Cao, Siyu Li, Yixin Liu, Zhiling Yan, Yutong Dai, Philip S. Yu, and Lichao Sun. A comprehensive survey of ai-generated content (aigc): A history of generative ai from gan to chatgpt, 2023.

[4] Christopher D Chambers and Loukia Tzavella. The past, present and future of registered reports. *Nature human behaviour*, 6(1):29–42, 2022.

[5] Allen Cypher, editor. *Watch What I Do: Programming by Demonstration*. MIT Press, 1993.

[6] Werner Damm and David Harel. Lscs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[7] Christof Ebert and Panos Louridas. Generative ai for software practitioners. *IEEE Software*, 40(4):30–38, 2023.

[8] Pelle Ehn. *Work-oriented Design of Computer Artifacts*. Arbetslivscentrum, 1988.

[9] Eric Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[10] Roberto Gozalo-Brizuela and Eduardo C. Garrido-Merchan. Chatgpt is not all you need. a state of the art review of large generative ai models, 2023.

[11] Kshitij Gupta. Llms can iteratively design, implement, and debug code with external tools! *https://kshitijkg.github.io/blog*, 4 2023.

[12] Maanak Gupta, CharanKumar Akiri, Kshitiz Aryal, Eli Parker, and Lopamudra Praharaj. From ChatGPT to ThreatGPT: Impact of generative ai in cybersecurity and privacy. *IEEE Access*, 2023.

[13] David Harel, Guy Katz, Rami Marelly, and Assaf Marron. Wise computing: toward endowing system development with proactive wisdom. *Computer*, 51(2):14–26, 2018.

[14] David Harel and Rami Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, 2003.

[15] David Harel and Rami Marelly. Specifying and executing behavioral requirements: the play-in/play-out approach. *Software & Systems Modeling*, 2:82–107, 2003.

[16] David Harel, Assaf Marron, and Gera Weiss. Behavioral programming. *Communications of the ACM*, 55(7):90–100, 2012.

[17] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review, 2023.

[18] Mark Ingebretsen. In the news. *IEEE Intelligent Systems*, 25(4):4–8, Jul 2010.

[19] Harshit Joshi, José Cambronero Sanchez, Sumit Gulwani, Vu Le, Gust Verbruggen, and Ivan Radiček. Repair is nearly generation: Multilingual program repair with LLMs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 5131–5140, 2023.

[20] Barbara Ann Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 07 2007.

[21] Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. TACO: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023.

[22] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *arXiv e-prints*, pages arXiv–2305, 2023.

[23] Brenda M Michelson. Event-driven architecture overview. `https://www.omg.org/news/whitepapers/EDA.pdf`, 2006.

[24] Bianca M. Napoleão, Fabio Petrillo, and Sylvain Hallé. Open source software development process: A systematic review. In *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 135–144, 2020.

[25] Bonnie A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, 1993.

[26] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, Inc., 2015.

[27] OpenAI. ChatGPT. https://chat.openai.com/.

[28] OSOCO. Contestia: Application tracking system. `https://contestia.es/`, 2023. Accessed: February 15, 2024.

[29] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. Mutation testing advances: an analysis and survey. In *Advances in Computers*, volume 112, pages 275–378. Elsevier, 2019.

[30] Oscar Pastor, Sergio España, José Ignacio Panach, and Nathalie Aquino. Model-driven development. *Informatik-Spektrum*, 31:394–407, 2008.

[31] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. The impact of AI on developer productivity: Evidence from GitHub Copilot. *arXiv preprint arXiv:2302.06590*, 2023.

[32] Klaas-Jan Stol and Brian Fitzgerald. The ABC of software engineering research. *ACM Trans. Softw. Eng. Methodol.*, 27(3), September 2018.

[33] Jiao Sun, Q Vera Liao, Michael Muller, Mayank Agarwal, Stephanie Houde, Kartik Talamadupula, and Justin D Weisz. Investigating explainability of generative ai for code through scenario-based design. In *27th International Conference on Intelligent User Interfaces*, pages 212–228, 2022.

[34] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[35] Vaughn Vernon. *Implementing domain-driven design*. Addison-Wesley, 2013.

[36] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. Software testing with large language model: Survey, landscape, and vision, 2023.

[37] Roel Wieringa. Design science as nested problem solving. In *Proceedings of the 4th international conference on design science research in information systems and technology*, pages 1–12, 2009.

[38] Terry Winograd. *Bringing Design to Software*. ACM Press/Addison-Wesley Publishing Co., 1996.

[39] Tianyu Wu, Shizhu He, Jingping Liu, Siqi Sun, Kang Liu, Qing-Long Han, and Yang Tang. A brief overview of chatgpt: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica*, 10(5):1122–1136, 2023.