



Software engineering department

Braude College of Engineering

OPM Code Generator

Project code: 25-2-D-15

Capstone Project Phase A – 61998

June 2025

Sagi Yosofov

Liroy Ben Shimon

Git repository link:

<https://github.com/SagiYosofov/OpmCodeGenerator.git>

Supervisor:

Dr. Natali Levi

Table of Contents

Abstract	4
<u>1 Introduction</u>	4
<u>2 Background and Related Work</u>	6
<u>2.1 The V-Model</u>	7
<u>2.1.1 Strengths</u>	7
<u>2.1.2 Weaknesses</u>	8
<u>2.2 SysML</u>	8
<u>2.2.1 Strengths</u>	8
<u>2.2.2 Weaknesses</u>	9
<u>2.3 OOSEM</u>	9
<u>2.3.1 Strengths</u>	9
<u>2.3.2 Weaknesses</u>	10
<u>2.4 xUML</u>	10
<u>2.4.1 Strengths</u>	10
<u>2.4.2 Weaknesses</u>	10
<u>2.5 fUML</u>	11
<u>2.5.1 Strengths</u>	11
<u>2.5.2 Weaknesses</u>	12
<u>2.6 MBSE</u>	12
<u>2.6.1 Strengths</u>	12
<u>2.6.2 Weaknesses</u>	13
<u>3 Expected Achievements</u>	13
<u>3.1 Criteria for Success</u>	14
<u>4 Engineering Process</u>	15
<u>4.1 Development methods</u>	15
<u>4.1.1 Python for the Backend</u>	15

<u>4.1.2 React for the Frontend</u>	15
<u>4.2 Our Learning Process of OPM</u>	15
<u>4.3 AI Learning Process of OPM</u>	16
<u>4.4 Claude: The Leading Choice for Learning OPM</u>	16
<u>4.5 From OPM Model to Code</u>	17
<u>5 Product</u>	18
<u>5.1 Architecture</u>	18
<u>5.2 System Flow</u>	19
<u>5.3 FR & NFR Requirements</u>	19
<u>5.3.1 Functional Requirements</u>	19
<u>5.3.2 Non-Functional Requirements</u>	20
<u>5.4 Use Case</u>	21
<u>5.5 User Interface</u>	22
<u>6 Testing Plan</u>	25
<u>6.1 Scope</u>	25
<u>6.2 Objectives</u>	25
<u>6.3 Test Constraints and Assumptions</u>	25
<u>6.4 Description of the Test Environment</u>	26
<u>6.5 Test Cases</u>	26
<u>7 References</u>	28
<u>8 AI Prompts</u>	29

Abstract

The OPM Code Generator project addresses the persistent gap between system specification and software implementation. Traditional development processes often suffer from communication breakdowns between specification and development stages, leading to misaligned systems requiring costly corrections. This project presents an innovative solution using artificial intelligence to automatically translate Object-Process Methodology (OPM) models into initial implementation in code. By transforming formal system models directly into code skeletons, software engineers can begin with an active code base aligned with original specifications rather than abstract documentation. This approach reduces development time, minimizes interpretation errors, and improves software quality. The project contributes to Model-Based Systems Engineering by demonstrating how AI systems can understand formal modeling languages and generate implementation code, bridging the critical gap between design and development phases.

Abbreviations

MBSE - Model-based systems engineering
OPM - Object-Process Methodology
OPL - Object-Process Language
SysML- Systems Modeling Language
UML - Unified Modeling Language
AI - Artificial Intelligence
OOSEM – Object Oriented Systems Engineering Method
xUML – Executable UML
fUML – Foundational UML
OMG - Object Management Group

1. Introduction

In the modern software development paradigm, a persistent challenge exists: the misalignment between customer requirements and the actual specification and development of systems. The traditional specification process begins when a systems analyst translates customer requirements into a specification document. This document is intended to serve as the foundation for software

engineers during the development phase. However, in practice, significant communication gaps exist between these stages.

Software engineers struggle to fully comprehend specification documents. In many cases, specification documents are either neglected or only partially addressed. The result is a system that fails to align with the original customer requirements. Therefore, numerous costly corrections are required in later development stages [1].

The cost of error correction increases exponentially as one progresses through the system's lifecycle stages [1]. According to the Model Fidelity Hierarchy, there exist several levels of precision in models, ranging from verbal concepts, through conceptual models and computational models, to executable models [1]. The transition between these levels enables the detection of various errors at each stage; however, these transitions typically involve information loss and require substantial effort.

The objective of this project is to develop a system based on artificial intelligence technologies that will help to bridge the gap between specification and development. The system will accept as input an OPM model representing the system specification and generate from it a basic software implementation that will serve as a starting point for software engineers. Through this approach:

- Software engineers can begin with an active code base rather than an abstract specification document.
- Better alignment between specification and implementation can be ensured.
- Development time will be reduced and the quality of the final product will improve.

In addressing the challenge of bridging the gap between system specification and software development, several methodologies and frameworks have emerged, each offering unique approaches to mitigate this divide. Prominent among these are the V-Model, SysML, OOSEM, xUML, fUML, and Model-Based Systems Engineering (MBSE). These solutions focus on enhancing traceability, improving communication, and enabling early validation throughout the development lifecycle, yet they exhibit various strengths and weaknesses that can affect their overall effectiveness in different contexts [2][4][5][7][8][10]. Within the scope of this project, we will examine the capability of advanced AI models to learn and process OPM diagrams describing the structure and operation of the system.

The primary stakeholders in this project are software development professionals involved in translating specification documents into code. The project is expected to impact the entire development process, from the specification stage

to final implementation, and to assist in closing the traditional gap between specification and actual development.

The innovation of this project lies in the integration of artificial intelligence technologies into the MBSE process, focusing on the critical gap between the specification stage and the development stage. Unlike existing approaches that focus on improving the specification process or separate development tools, the project offers an integrative approach that connects the stages through a unified and continuous model [1].

The project is expected to contribute to understanding how AI systems can be taught a "new language" - in this case, a formal modeling language like OPM and use this knowledge to streamline the entire development process and improve the quality of the resulting systems.

The structure of this document is divided into several chapters, each corresponding to a different stage of the project. Following the introduction, Chapter 2 presents a comprehensive literature review and an overview of existing methods and tools aimed at addressing the identified problem. Chapter 3 outlines the expected achievements of the project, including the deliverables to be developed, along with a description of their functionality and clear success criteria for evaluating their effectiveness. Chapter 4 describes the engineering process: the development process, including the rationale behind the chosen approach and identified constraints. Chapter 5 describes the product itself, detailing models, and user interface design. Finally, chapter 6 includes the testing plan, outlining the methods, assumptions, and environment used for system validation.

2. Background and Related Work

The gap between system design and development represents a longstanding challenge in software engineering. Various methodologies and frameworks have been introduced to mitigate this divide, each offering distinct approaches to facilitating the transition from system specification to implementation.

Prominent solutions such as the V-Model, SysML, OOSEM, xUML, fUML, and Model-Based Systems Engineering (MBSE) emphasize key concepts including traceability, early validation, and integration across different phases of development. While these models have demonstrated their utility in enhancing the alignment between design and development processes, they also exhibit limitations, particularly in terms of implementation complexity, adaptability to changing requirements, and the consistency between design models and executable code. This literature review critically examines these frameworks,

assessing their strengths, weaknesses, and overall effectiveness in bridging the specification-development gap in complex systems engineering contexts.

2.1 The V-Model

The V-Model represents one of the principal solutions for bridging the gap between specification and software development. The model depicts the system development process as a "V" where the left branch represents the decomposition and definition of requirements, while the right branch represents the integration and verification phases. Unlike the classic Waterfall model, in the V-Model verification planning occurs concurrently with requirements definition, with each level on the left side corresponding to a parallel level on the right side. The V-Model creates an efficient bridge between specification and development through several complementary mechanisms. The model establishes a direct relationship between requirements and verification, such that each level of requirements has a specific verification method defined for it, ensuring that every requirement is measurable and verifiable, thereby reducing the gap between the specification and the ability to confirm that the development meets the requirements [2]. Continuous integration of systems engineering is manifested in emphasizing the role of the systems engineer throughout the entire lifecycle, with a clear division of responsibilities between systems engineering and development engineering, as demonstrated in the V-Model diagram through the line separating areas of responsibility [3].

2.1.1 Strengths

The V-Model offers several notable strengths in bridging the gap between system specification and development. One of its key advantages is traceability, as the model creates a clear link between requirements and implementation, ensuring a logical sequence of activities throughout the system lifecycle [2]. Additionally, the V-Model enhances communication by providing a visual and intuitive representation of the development process, which improves communication among stakeholders [3]. The model also facilitates early error detection, as the parallel verification process allows issues to be identified earlier in development, significantly reducing the cost of corrections [2]. Another strength is its promotion of concurrent engineering, where domain experts (e.g., security, user interface, manufacturing, and maintenance specialists) are integrated early in the concept phase, minimizing costly changes later in the project [3].

2.1.2 Weaknesses

The V-Model presents several weaknesses that may limit its effectiveness in certain contexts. One of the primary challenges is its complexity in implementation, as the model requires a deep understanding of each of its stages, particularly when dealing with large systems. For teams with limited experience, implementing the model can be challenging and may require significant effort in learning and adaptation [2]. Furthermore, although the model supports iterations, it is less suited to environments with frequent requirement changes. The V-Model recommends freezing user requirements after the Preliminary Design Review (PDR), which means significant changes may only be incorporated in the next version of the system, posing a challenge in dynamic environments [3]. The model also places a significant emphasis on documentation, requiring formal definitions at each stage, which can be burdensome for smaller projects or those that require rapid response times [2]. Additionally, the model is less compatible with pure agile development, as it is less suited to methods that rely on very short iterations and minimal upfront planning, as seen in agile frameworks [3]. Finally, due to its visual similarity to the Waterfall model, there is a risk that teams may mistakenly interpret the V-Model as a rigid linear process, despite its support for iterations and bidirectional progression along the "V" [2].

2.2 SysML

The Systems Modeling Language (SysML) is a modeling language based on UML that provides a structured and graphical way to document and model requirements from the early stages of system specification. The model utilizes Requirements Diagrams, which help organize requirements, show the relationships between them, and define clear semantics. Additionally, SysML enables linking these requirements to other system models, such as Use Case Diagrams, thereby addressing the traditional gap between requirement specification and system design and implementation [4].

2.2.1 Strengths

One of the key advantages of SysML is its ability to represent requirements graphically through diagrams, rather than relying solely on textual descriptions. This visual approach makes requirements an integral part of the system's architecture. The graphical modeling improves communication between all stakeholders involved in the project. Furthermore, the model provides

mechanisms for identifying and managing relationships between requirements themselves and between the requirements and other system models, significantly enhancing the traceability of requirements and easing their maintenance [4].

2.2.2 Weaknesses

One potential drawback of SysML is the need for learning and implementing a new modeling language within an organization, which requires time and training for the team. Additionally, the model is less suited for dynamic environments with frequent changes in requirements, similar to agile development methodologies, and is better suited for environments with more stable requirements [4]. Furthermore, the visual similarity of the Requirements Diagram to the Waterfall model may lead to the mistaken impression that it represents a rigid, linear process, despite the fact that the model actually allows for iterative flexibility between stages [4].

2.3 OOSEM

The Object-Oriented Systems Engineering Method (OOSEM) is a methodology that integrates object-oriented principles with traditional systems engineering approaches. Its primary goal is to bridge the traditional gap between systems engineering and software engineering. By employing graphical models and object-oriented techniques throughout the system lifecycle, OOSEM facilitates closer collaboration between systems engineers and software engineers, beginning in the early stages of system specification and extending through implementation and verification [5].

2.3.1 Strengths

One of the notable strengths of OOSEM is its ability to leverage object-oriented techniques to address the increasing complexity of modern systems. The graphical models provided by SysML (e.g., requirements diagrams, behavior diagrams, and structure diagrams) allow for a gradual and consistent description of the system, with increasing levels of detail, while reusing well-defined building blocks. Furthermore, the methodology places significant emphasis on validation and verification at every stage of development. Early simulations and virtual prototypes of the system reduce risks and enable early identification and resolution of issues [5].

2.3.2 Weaknesses

One of the challenges in implementing OOSEM is the need for extensive training for teams in the methodology and the associated tools, particularly if the organization is unfamiliar with object-oriented development principles. Without a solid understanding of the approach and how it integrates systems engineering and software engineering, challenges may arise in adopting and implementing the methodology. Additionally, the numerous graphical models used in OOSEM may be perceived as cumbersome and unintuitive, particularly for non-technical stakeholders, when compared to traditional text-based documentation [5].

2.4 xUML

The xUML (Executable UML) model is a profile of the UML modeling language that enables the automatic generation of executable code directly from UML models. By utilizing standard UML diagrams, such as class and state diagrams, along with an Action Language that defines system behavior, xUML allows for a seamless translation from high-level design to executable code. This capability ensures consistency between the design and implementation phases, bridging the traditional gap between specification, design, and development. The models created in xUML can be tested and validated early in the development process, enabling early detection of potential issues and reducing the risk of discrepancies between the design and the final implementation [7].

2.4.1 Strengths

One of the main advantages of xUML is its ability to produce executable and testable models early in the project, allowing for early validation of design and reducing risks and costs. The use of UML preserves high-level abstraction in the models, without the need for implementation details. The capability to translate models directly into code shortens development time and prevents duplication and inconsistencies between models and code. Additionally, the models can serve as a foundation for implementation using the Model-Driven Architecture (MDA) approach [6].

2.4.2 Weaknesses

Some challenges in implementing xUML include the need for extensive training

of the team in using the profile and associated tools, as well as the integration of new development processes within the organization. Furthermore, the Action Language may be limited and not provide all the capabilities of full-fledged programming languages [7]. Another difficulty is expressing concurrent and time-dependent behavior using state machines. Finally, the models still require either manual or automatic mapping to execution code, which creates a risk of discrepancies between the model and the implementation [6]. Furthermore, this solution requires to divide the system into several subsystems that have little or no interaction between them [2].

2.5 fUML

fUML (Foundational UML) is an Object Management Group (OMG) standard that defines formal semantics for the behavioral elements of UML, providing a central solution to bridge the gap between software design and development. The model offers a precise specification for an execution engine for UML models, defining how the models should behave at runtime [8]. Unlike standard UML, which only provides graphical syntax, fUML adds formal semantic meaning that enables the execution of the models. The primary advantage of this approach is that it allows models to transform from passive descriptions into active working products, enabling system simulation and execution during the design phase, long before actual development. This approach helps bridge the gap between design and development [9].

2.5.1 Strengths

The primary strengths of fUML as a solution for bridging design and development include its extensibility, which enables customization for specific domains through UML's profile mechanism. The model is built in a modular way based on the Visitor design pattern, allowing the addition or expansion of semantic visitors that define the behavior of model elements [8]. Another significant advantage is its ability to integrate with external tools. This capability enables models to be linked to existing systems and integrated with other software tools [8]. Furthermore, the model supports observation and control of execution, allowing for the verification of model correctness during the design phase through execution, monitoring, and debugging [8].

2.5.2 Weaknesses

Despite its advantages, fUML has several significant limitations. First, the model does not directly address the concept of time, which complicates the modeling of real-time systems without additional extensions [8]. Second, the model encounters difficulties when integrating multiple profiles, where a model component has several stereotypes applied to it, each with its own semantics. While the semantics of these stereotypes can be captured in model extensions, it is currently not possible to integrate different visitors at runtime to interpret a single model element [8]. Additionally, the current implementation of fUML requires significant expertise to identify points where control must be transferred from the semantic engine to external tools, which complicates its extension by developers who are not experts in the model [8]. Another challenge is the need to rewrite existing code due to inconsistencies in the use of the Factory pattern, which results in code duplication and complicated maintenance [8]. Furthermore, this solution requires dividing the system into several subsystems that have little or no interaction between them [2].

2.6 MBSE

Model-Based Systems Engineering (MBSE) represents a significant methodological approach to bridging the gap between the specification and development stages of software systems. In contrast to the traditional document-based approach, MBSE offers a workflow where the model serves as the central and authoritative source of truth. The formal model provides a shared and unambiguous language for all stakeholders in the project. The approach includes a Model Fidelity Hierarchy, which progresses from written text, through conceptual models, computational models, to executable models [1]. The continuous transition between these stages, utilizing the same modeling language (such as OPM), ensures consistency and completeness of requirements and design, thereby preventing the loss of critical information between different development stages.

2.6.1 Strengths

The primary strength of MBSE in bridging the gap between design and development is its integrative approach, which enables early detection of errors

in the development process. Identifying issues at early stages significantly reduces the cost of corrections, which tends to increase exponentially as the system lifecycle progresses [1]. The model allows for the gradual integration of both qualitative (conceptual) and quantitative (computational) elements, creating synergies that reveal errors and ambiguities that cannot be detected when examined independently. A unified modeling language enables effective communication among various stakeholders, ranging from project managers, system engineers, developers, to testing teams, adapting the level of detail and perspective to meet the specific needs of each group [10]. Additionally, the ability to run simulations and visualizations enables early validation of designs before the actual development phase begins, thus avoiding unnecessary and costly development cycles.

2.6.2 Weaknesses

MBSE has several limitations in bridging the gap between design and development. First, the learning curve for MBSE tools is steep, requiring a significant investment in training teams and integrating the processes [1]. Second, although MBSE promises a smooth transition between the various stages of modeling, in practice, integrating conceptual models with executable code is particularly challenging, especially in complex systems. There are also difficulties in supporting all computational and executional aspects within a single modeling language [1]. Moreover, even when using the MBSE approach, errors may still be discovered during the execution phase, such as initialization errors or formula inconsistencies [1]. Finally, while the approach promises continuity and completeness, it does not address all communication issues between specification and development teams, especially in areas such as understanding the business context or requirements that are difficult to express in a formal model.

3 Expected Achievements

- **Automated Code Skeleton Generation from OPM Models:** The primary outcome of this project is the development of a system capable of accepting an OPM model as input, representing the system's specification and automatically generating a basic code skeleton. This generated code will serve as a starting point for software engineers, enabling them to begin development from an active code base rather than an abstract specification document. This outcome is expected to

significantly reduce development time, minimize ambiguity, and improve the alignment between system specifications and their implementation.

- **Evaluation of AI Comprehension of Visual and Textual Inputs:** The system will demonstrate the ability of advanced artificial intelligence models to learn and process both visual OPM diagrams and textual prompts instructing how to translate OPM to code. The findings will provide meaningful insights into how AI systems interpret formal modeling languages.
- **Development of a User Friendly System:** The system will include a simple and intuitive interface, making it easy for users to review, understand, and even edit the generated code. This will make software engineers development process faster, with a clear connection between the system design and the actual implementation.

3.1 Criteria for Success

- **Parsing Accuracy:** The AI model will correctly identify at least 85% of system elements (objects, processes, and relationships) from OPM diagrams.
- **Code Generation Validity:** The generated code will compile successfully and reflect the logic of the original OPM model in at least 80% of test cases.
- **User Satisfaction:** User satisfaction will be assessed through a structured survey distributed to systems engineers and software developers. The target is to achieve at least 80% satisfaction across five key dimensions:
 1. The quality of the generated code, measured in terms of correctness and architectural structure.
 2. Time savings in the development process.
 3. Ease of use of the system.
 4. Accuracy of the model-to-code translation.
 5. The system's contribution to improving communication between development teams.
- **System reliability & performance:** The system must work with a minimum uptime of 95%, maintaining good performance stability even under high load scenarios or when subjected to different conditions.

4 Engineering Process

4.1 Development methods

4.1.1 Python for the Backend

Python is widely used for backend web development due to its simplicity, flexibility, and suitability for rapid prototyping. The language benefits from a strong ecosystem, including lightweight frameworks such as Flask for building web APIs. Additionally, Python integrates seamlessly with artificial intelligence tools, enhancing its applicability in data-driven applications. These features collectively make Python an ideal choice for developing robust and scalable web services [11].

4.1.2 React for the Frontend

React is widely used for frontend development due to its component-based architecture, which supports modular and maintainable UI design. Its use of a virtual Document Object Model (DOM) enhances performance by enabling efficient updates and rendering. React components are reusable and declarative, simplifying testing and long-term maintenance. Furthermore, React aligns with modern web development practices by supporting Single Page Applications (SPAs), which offer dynamic content updates and a seamless user experience. These features collectively make React a robust choice for building scalable and responsive frontend interfaces [12].

4.2 Our Learning Process of OPM

Our learning process of OPM methodology focused on understanding the core theoretical principles. First, we studied the foundation of MBSE, where the conceptual model of a system is expressed as early as possible in the development process. Within OPM framework, we learned that every system is represented through just two basic components: objects (things that exist) and processes (things that affect objects).

We delved into the types of objects, physical and informative, and how they are visually represented in diagrams. We studied the various states that objects can

have, and the types of processes. We explored the types of procedural links (result, consumption, effect, agent, instrument, and event) and structural links (aggregation, exhibition, generalization, and classification). We also learned about hierarchical structure in models through the use of Inzoom and Unfold techniques, which allow detailed representation at different levels of processes and objects within the system.

4.3 AI Learning Process of OPM

In our teaching process of AI, we provided the AI system with OPM presentation that contains all the theoretical data it needs to know about OPM and practical exercises. Later, we provided the AI numerous practical prompts to ensure it truly understood the theoretical material. We asked it diverse questions about key concepts in OPM, and when it answered correctly, we gave positive reinforcing feedback that helped anchor the knowledge. Conversely, when it made errors in understanding or implementation we immediately provided corrective notes that helped refine its knowledge and correct misconceptions.

A significant part of the training included presenting visual examples of different model notations, to ensure the AI system recognized and understood the meaning of each - from marking physical and informative objects, through identifying different object states, to denote various procedural and structural links. Gradually, we progressed from simple examples to more complex ones involving many processes, objects, and relationships between them. We challenged the AI system with analyzing complete and complex systems to see if it could apply OPM principles in complex, realistic situations, and to test its ability to understand the hierarchies and multiple levels of abstraction that are typical in OPM models.

4.4 Claude: The Leading Choice for Learning OPM

We conducted a comprehensive comparison between several leading artificial intelligence models (ChatGPT, Gemini and Claude) with the aim of identifying the most suitable platform for learning OPM. After experimenting with each model, we found that Claude demonstrated a significantly superior ability to process complex information and theoretical explanations.

While ChatGPT exhibited a basic understanding of key concepts, it struggled with comprehending the intricate hierarchies and the relationships between objects and processes. Gemini, on the other hand, was effective at identifying visual elements but failed to integrate them coherently with theoretical

explanations. What set Claude apart was its impressive capacity to handle complex explanations, maintain consistency throughout extended discussions, and present a comprehensive understanding of the OPM model. Furthermore, Claude displayed a notable ability to learn from mistakes and gradually improve his understanding, which was crucial for our ongoing learning process. Its flexibility in dealing with diverse and complex prompts led us to choose Claude as our primary learning platform.

4.5 From OPM Model to Code

As part of training our model, we developed a structured and systematic approach for transforming OPM models into Python code. This process involved designing a series of focused prompts, each detailing how to convert specific elements of the model into their corresponding code representations. For each of the core components: objects, processes, and relationships, we defined prompts that effectively guide the translation into Python.

For example, regarding objects, we trained the model to translate OPM objects into Python classes. For processes, we trained Claude to define them as methods in the class that represents the object that is connected to them with agent link. A method that represents a process will get as parameters all the objects that are connected to the process with instrument link.

Various types of relationships such as aggregation, generalization, classification and influence were addressed through detailed implementation guidelines. These instructions made use of relevant programming techniques, including inheritance, composition etc. Each prompt was accompanied by concrete code examples illustrating the preferred implementation style, enabling the model to learn the nuances of each type of transformation. This methodical approach enables our system to translate complex OPM diagrams into executable Python code while preserving the original semantics and design principles of the model. As a result, the generated code adheres to the defined standards and offers a practical, structured solution to model-driven development.

5. Product

5.1 Architecture

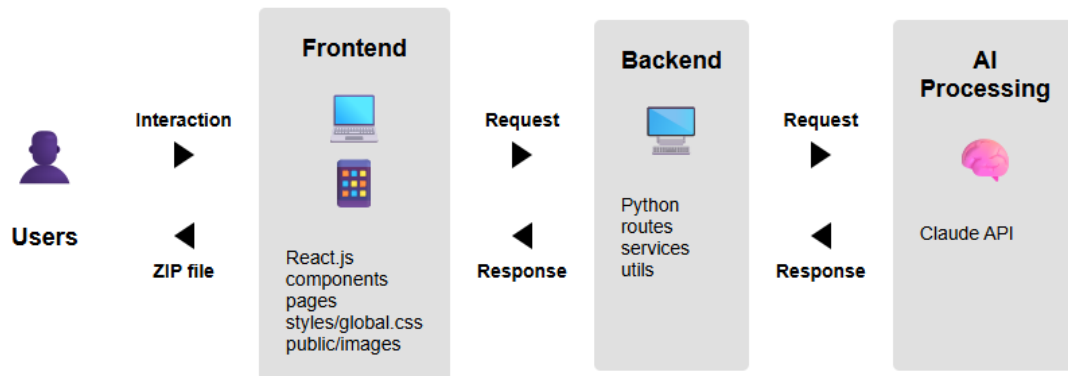


Figure 1: Architecture Diagram

The final architecture of the system consists of three main components that communicate efficiently and modularly:

1. **User Interface (Frontend)** - Built on React.js, the system provides an intuitive interface that allows users to upload OPM diagrams, select programming language and download the generated code. The interface includes components, pages, styling (styles/global.css), and graphical assets (public/images), and is adapted for both computers and mobile devices. Users interact with the system through this interface and receive a downloadable ZIP file that contains all the code files.
2. **Server (Backend)** - Based on Python, the server manages the business logic of the system. It contains API routes, data processing services, and utility functions. The server receives requests from the interface, processes them, and communicates with the AI service for diagram analysis when needed. After receiving the results, it returns processed responses to the user interface.
3. **Artificial Intelligence Processing (AI Processing)** - Based on the Claude API, this component forms the core of the system. It receives requests from the server with OPM diagrams for analysis, decodes the structures and relationships in the diagrams using advanced language models, and generates appropriate code. The results are returned to the server as structured responses.

5.2 System Flow

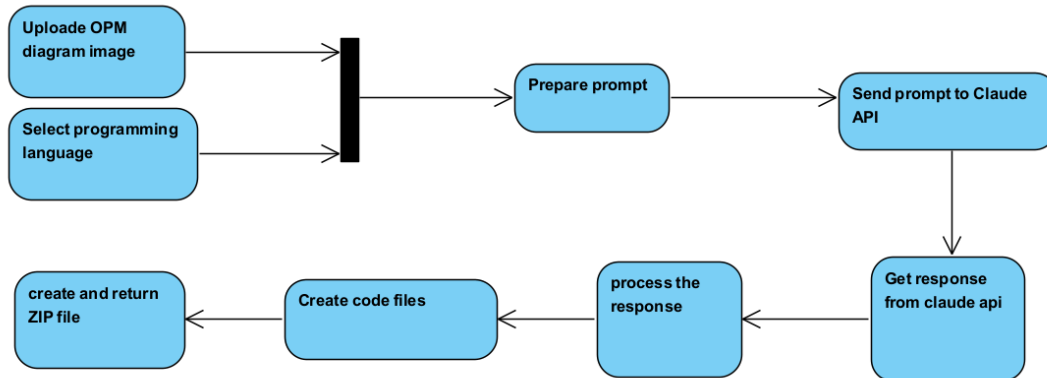


Figure 2: System Flow Diagram

The diagram illustrates the workflow of the system for converting OPM diagrams to code. The process begins with two user actions: uploading an OPM diagram image and selecting the desired programming language. The system then prepares a structured prompt that incorporates both the diagram and language specifications, which is sent to the Claude API. Upon receiving a response from the API, the system processes the returned content and analyzes it to identify various software components. Based on this analysis, the system generates appropriate code files in the specified structure and language. Finally, all files are packaged into a single ZIP file that is returned to the user, containing the complete project structure with all code files created according to the original diagram.

5.3 FR & NFR Requirements

5.3.1 Functional Requirements

No.	Requirement
1	The system shall allow uploading an image.
2	The system shall allow selecting a programming language.
3	The system shall allow translating an OPM model into code.
4	The system shall allow integration with the Claude API.
5	The system shall allow generating code files.
6	The system shall allow writing documentation for the generated code.
7	The system shall allow creating a ZIP file.
8	The system shall allow downloading the ZIP file.

5.3.2 Non-Functional Requirements

No.	Requirement	Type
1	The system shall support image uploads in JPG, JPEG, and PNG formats only.	Compliance
2	The programming language options including Python, Java, C#, and C++.	Compliance
3	The generated code files are with the following extensions: .py, .java, .cs, and .cpp.	Compliance
4	The system shall be responsive and adapt to various screen sizes and devices.	Usability
5	The system shall provide a user-friendly interface that is intuitive and easy to navigate.	Usability
6	The system shall support concurrent usage by at least 10 users without performance degradation.	Performance
7	The system shall allow image uploads of up to 5 MB per file.	Performance
8	The system shall return a response within 2 minutes of the time a request is submitted.	Performance
9	The system shall be easy to maintain and update, using modular architecture and providing clear documentation.	Maintainability
10	The system shall be available at least 95% of the time each calendar month, excluding scheduled maintenance.	Availability

5.4 Use Case

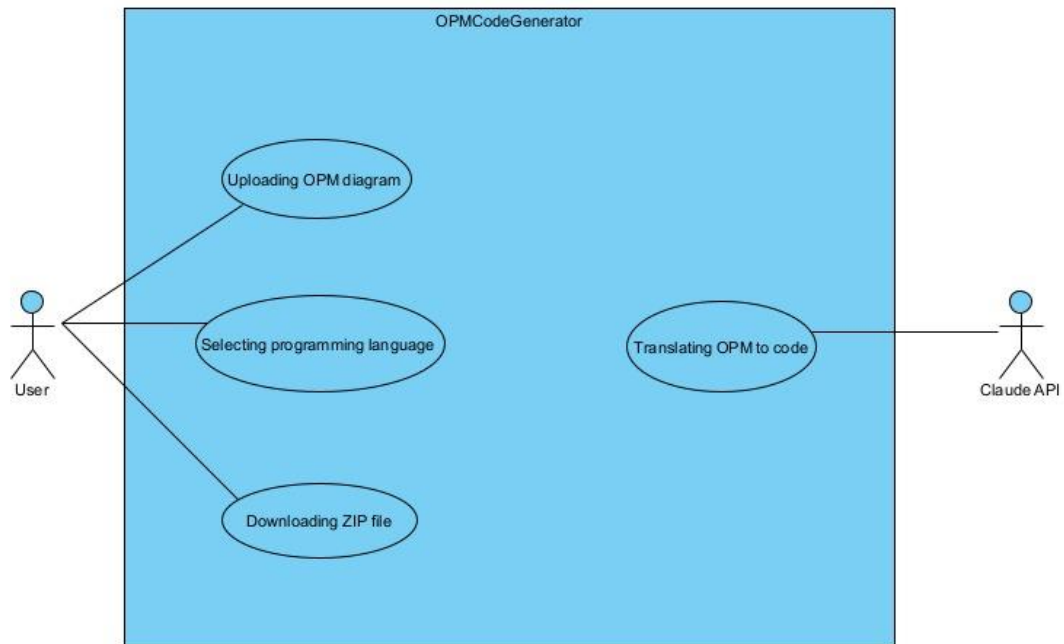
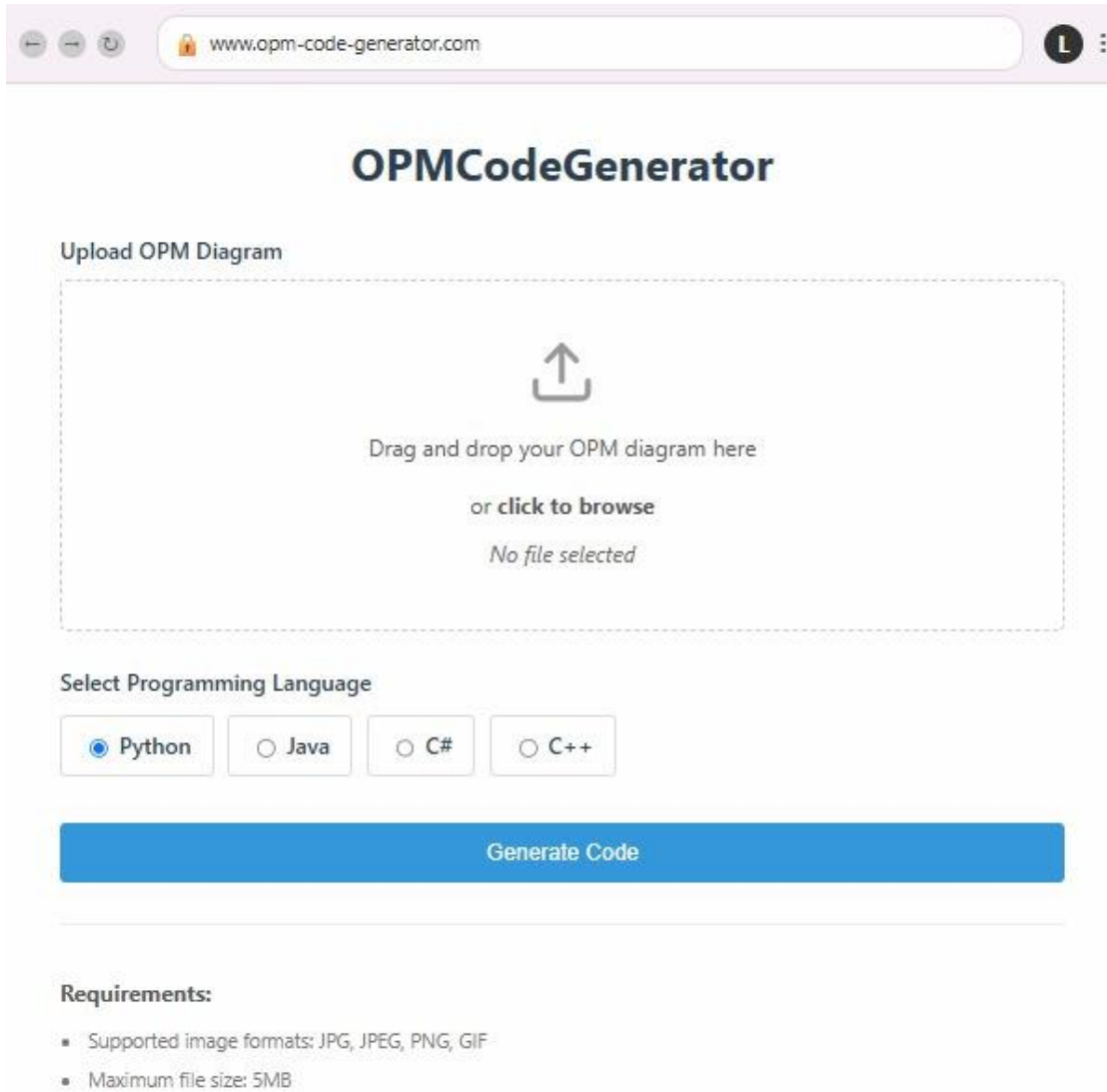


Figure 3: Use Case Diagram

5.5 User Interface



The screenshot shows a web browser window with the address bar displaying "www.opm-code-generator.com". The page title is "OPMCodeGenerator". Below the title, there is a section titled "Upload OPM Diagram". Inside this section, there is a large dashed box containing an upload icon (an upward arrow inside a square) and the text "Drag and drop your OPM diagram here" and "or click to browse". Below this, it says "No file selected". Below the upload section, there is a section titled "Select Programming Language". This section contains four radio buttons: "Python" (selected), "Java", "C#", and "C++". Below these options is a large blue button labeled "Generate Code". At the bottom of the page, there is a section titled "Requirements:" which lists two items: "Supported image formats: JPG, JPEG, PNG, GIF" and "Maximum file size: 5MB".

OPMCodeGenerator

Upload OPM Diagram

Drag and drop your OPM diagram here
or click to browse
No file selected

Select Programming Language

☒ Python ☐ Java ☐ C# ☐ C++

Generate Code

Requirements:

- Supported image formats: JPG, JPEG, PNG, GIF
- Maximum file size: 5MB

Figure 4: Screen 1, OPM diagram loading and programming language selection.

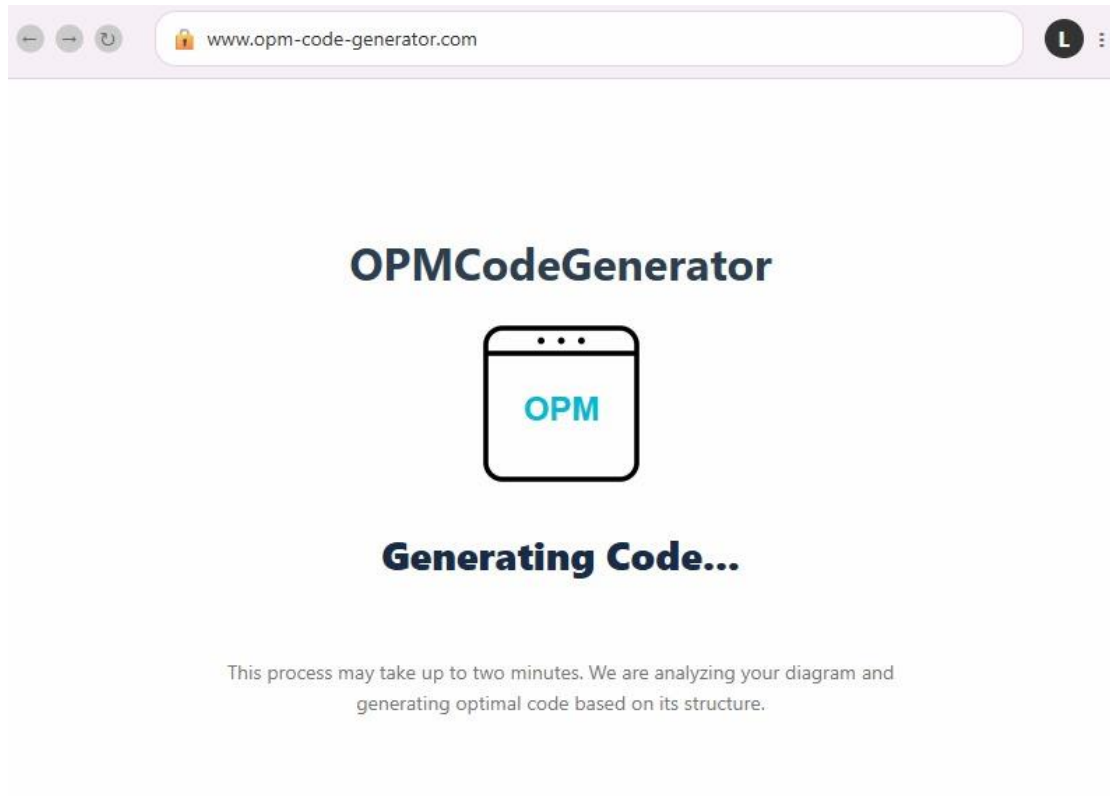
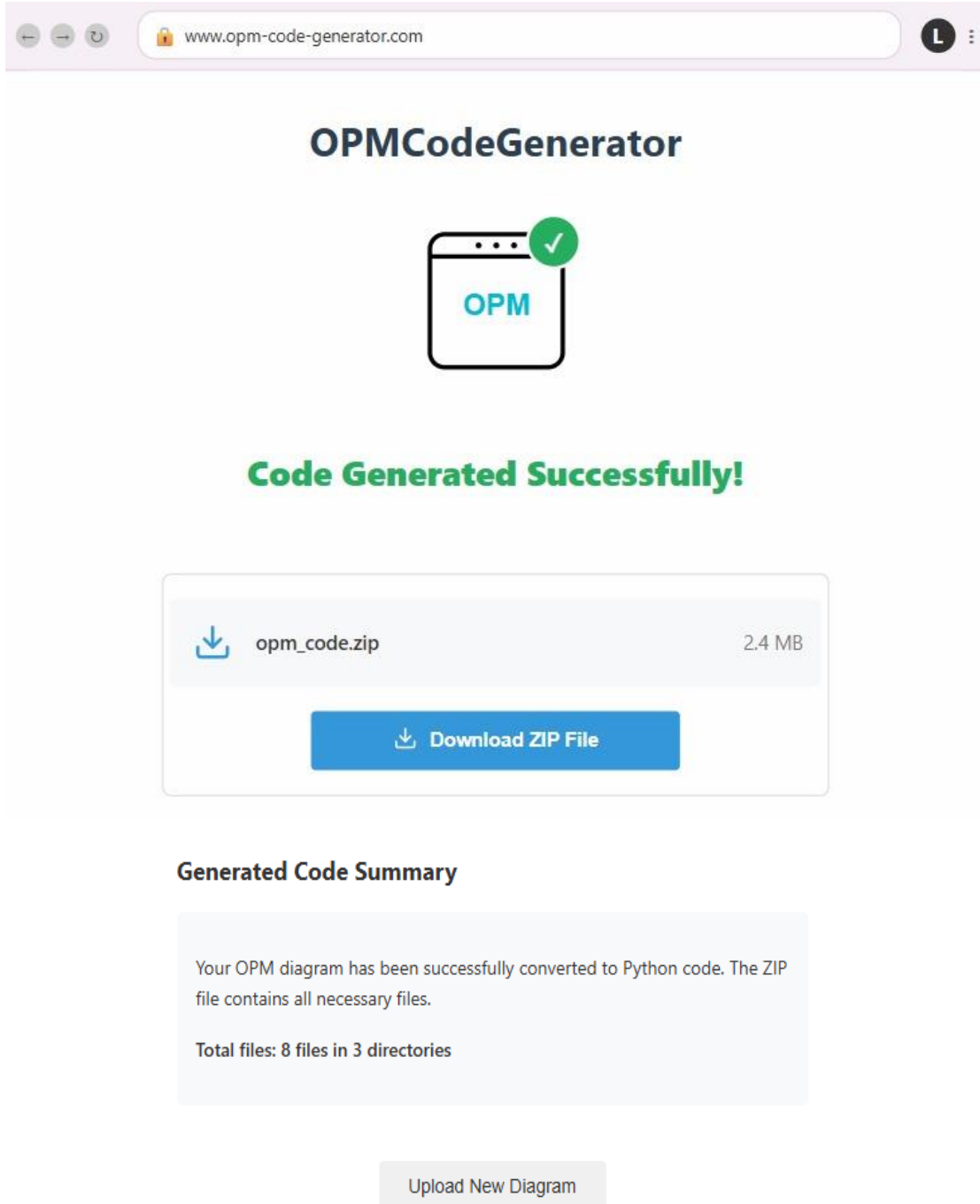


Figure 5: Screen 2, user see that the system process the input and generating results.



The screenshot shows a web browser window with the address bar displaying "www.opm-code-generator.com". The page title is "OPMCodeGenerator". Below the title is a logo consisting of a square with "OPM" inside and a green checkmark in the top right corner. The main heading is "Code Generated Successfully!". Below this, there is a download section showing a file named "opm_code.zip" with a size of "2.4 MB" and a blue "Download ZIP File" button. Underneath is a "Generated Code Summary" section with the text: "Your OPM diagram has been successfully converted to Python code. The ZIP file contains all necessary files." and "Total files: 8 files in 3 directories". At the bottom is a button labeled "Upload New Diagram".

OPMCodeGenerator

OPM

Code Generated Successfully!

opm_code.zip 2.4 MB

Download ZIP File

Generated Code Summary

Your OPM diagram has been successfully converted to Python code. The ZIP file contains all necessary files.

Total files: 8 files in 3 directories

Upload New Diagram

Figure 6: Screen 3, system display results.

6. Testing Plan

6.1 Scope

This test plan covers the functional and non-functional testing of the OPM-to-Code Generation System. The scope includes verifying the system's capabilities to accept input in the form of OPM images, interact with external APIs (Claude), generating code in multiple languages, and package the code into a ZIP file. It also includes performance, usability, and compliance tests to ensure the system operates within defined parameters.

6.2 Objectives

- Ensure all functional and nonfunctional requirements are met.
- Verify system performance and responsiveness.
- Validate usability and user experience.
- Validate the integration with Claude API.
- Confirm the correctness and structure of the generated source code files.

6.3 Test Constraints and Assumptions

- The system will have internet access.
- Testing will be limited to the listed programming languages: Python, Java, C#, and C++.
- Image recognition and OPM understanding will rely on Claude capabilities.
- Users will not upload images larger than 5 MB, and only supported formats (JPG, JPEG, PNG) will be tested.
- Claude API and AI services are assumed to be available and functioning during testing.

6.4 Description of the Test Environment

- **Frontend:** React.
- **Backend:** Python.
- **AI Service:** Anthropic's Claude 3.7 Sonnet AI assistant service.
- **Storage:** Local file system for uploaded images and generated files.
- **Browser:** Latest versions of Chrome, Firefox, and Edge.
- **Tools:** Playwright (UI automation) and PyTest (unit testing).

6.5 Test Cases

No.	Name	Description	Input	Procedure	Expected Result
1	valid_image_upload	Upload image in supported format and size	Image: opm.jpeg	1. Click "Upload OPM diagram". 2. Choose opm.jpeg file.	Upload successful
2	Unsupported_format_image_upload	Upload image in unsupported format	Image: opm.txt (Size- 1MB)	1. Click "Upload OPM diagram". 2. Choose opm.txt file.	Error message displayed: "Uploaded image format is not supported "
3	large_image_upload	Upload image > 5MB	Image: opm.jpeg (Size- 6MB)	1. Click "Upload OPM diagram". 2. Choose opm.jpeg file.	Error message displayed: " Maximum image size is 5MB "
4	language_selection	Selecting programming language	Language: Pyhton	1. Select 'Python' from the radio buttons	Python selected successfully.
5	valid_code_generation	Generate code from OPM image	Image: Opm.jpeg Language: Java	1. Upload Opm.jpeg. 2. Select Java. 3. Click "Generate Code".	Java code files are generated
6	valid_code_generation_language_not_selected	Generate code from OPM image	Image: Opm.jpeg	1. Upload Opm.jpeg. 2. Click "Generate Code".	Python code files are generated (python is default selection)

7	invalid_code_generation_image_not_uploaded	Generate code without uploading opm image.	Language: Java	1. Select Java. 2. Click "Generate Code".	Error message: "Image not uploaded"
8	invalid_code_generation_image_does_not_contain_opm_diagram	Generate code from image that doesn't contain OPM diagram	Image: use_case.jpg Language: Java	1. Upload use_case.jpg. 2. Select Java. 3. Click "Generate Code".	Error message: "the uploaded image doesn't contain OPM diagram"
9	zip_creation	Package generated code into ZIP	Image: Opm.jpeg Language: Java	1. Upload Opm.jpeg. 2. Select Java. 3. Click "Generate Code".	ZIP file that contains the code files created.
10	zip_download	Download the generated ZIP file	Image: Opm.jpeg Language: Java	1. Upload Opm.jpeg. 2. Select Java. 3. Click "Generate Code". 4. Click "Download ZIP FILE".	ZIP is downloaded
11	ui_responsiveness_desktop	Check UI layout and behavior on desktop screen size	Desktop resolution (1920×1080)	1. Open the application on a desktop.	UI adapts and displays correctly
12	ui_responsiveness_tablet	Check UI layout and behavior on tablet screen size	Tablet resolution (768×1024)	1. Open the app on a tablet.	UI adapts and displays correctly
13	ui_responsiveness_mobile	Check UI layout and behavior on mobile screen size	Mobile resolution (e.g., 375×667)	1. Open the app on a mobile device.	UI adapts and displays correctly
14	Ten_concurrent_users	Test system with 10 users at once	-	1. Simulate 10 users - each uploads and generates code.	No performance issues
15	response_time	Ensure request completes in ≤ 2 min	Image: Opm.jpeg Language: java	1. Upload Opm.jpeg. 2. Select java. 3. Click "Generate Code".	Response in no more than 2 minutes
16	file_extension_validation	Ensure generated files match selected language	Image: Opm.jpeg Language: python	1. Upload Opm.jpeg. 2. Select Python. 3. Click "Generate Code".	Generated files end with .py and contain python code.
17	code_quality_review	Evaluate modularity and documentation	Image: Opm.jpeg Language: python	1. Upload Opm.jpeg. 2. Select python.	Clean structure and comments

				3. Click "Generate Code". 4. Review generated files.	
18	usability_testing	Verify the system provides an intuitive and easy to navigate interface	-	1. Conduct usability testing with 5 representative users. 2. Ask users to complete key tasks without assistance. 3. Collect feedback on ease of use – asking them to complete SUS survey.	SUS score ≥ 80

7. References

- [1] Levi-Soskin, N., Jbara, A., & Dori, D. (2020). The model fidelity hierarchy: From text to conceptual, computational, and executable model. *IEEE Systems Journal*, 15(1), 1287-1298.
- [2] Li, L., Soskin, N. L., Jbara, A., Karpel, M., & Dori, D. (2019). Model-based systems engineering for aircraft design with dynamic landing constraints using object-process methodology. *IEEE Access*, 7, 61494-61511.
- [3] Forsberg, K., & Mooz, H. (1991). The relationship of system engineering to the project cycle. *Center for Systems Management*, 5333.
- [4] dos Santos Soares, M., & Vrancken, J. L. (2008). Model-Driven User Requirements Specification using SysML. *J. Softw.*, 3(6), 57-68.
- [5] Pyster, A., Adcock, R., Ardis, M., Cloutier, R., Henry, D., Laird, L., ... & Wade, J. (2015). Exploring the relationship between systems engineering and software engineering. *Procedia Computer Science*, 44, 708-717.
- [6] Luz, M. P., & Da Silva, A. R. (2004, October). Executing UML Models. In *3rd Workshop in Software Model Engineering (WiSME 2004)* (Vol. 192).
- [7] Sulistyo, S., & Najib, W. (2002). Executable uml. *Dept of Information and Communications Technology Agder University College, Norway*.

- [8] Guerhazi, S., Tatibouet, J., Cuccuru, A., Dhouib, S., Gérard, S., & Seidewitz, E. (2015). Executable modeling with fuml and alf in papyrus: Tooling and experiments. *strategies*, 11, 12.
- [9] Tatibouët, J., Cuccuru, A., Gérard, S., & Terrier, F. (2014). Formalizing execution semantics of UML profiles with fUML models. In *Model-Driven Engineering Languages and Systems: 17th International Conference, MODELS 2014, Valencia, Spain, September 28–October 3, 2014. Proceedings 17* (pp. 133-148). Springer International Publishing.
- [10] Firesmith, D. (2003). Modern requirements specification. *Journal of Object Technology*, 2(2), 53-64.
- [11] Patkar, U., Singh, P., Panse, H., Bhavsar, S., & Pandey, C. (2022). Python for web development. *International Journal of Computer Science and Mobile Computing*, 11(4), 36.
- [12] Chen, S., Thaduri, U. R., & Ballamudi, V. K. R. (2019). Front-end development in react: an overview. *Engineering International*, 7(2), 117-126.

8. AI Prompts

- Learning process of Claude AI the OPM model.
<https://claude.ai/share/8a1429ff-181c-45ee-aac1-896fa47d38c1>
- Learning process of Claude AI how to translate OPM model into python code.
<https://claude.ai/share/f7a9e6e4-9847-42b2-a5a3-1ac3b1dfeaed>