Lessons Learned from the Use of Generative AI in Engineering and Quality Assurance of a WEB System for Healthcare

Guilherme H. Travassos
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, RJ, Brazil
ght@cos.ufrj.br

Sabrina Rocha
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, RJ, Brazil
sabrinarocha@cos.ufrj.br

Rodrigo Feitosa
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, RJ, Brazil
rfeitosa@cos.ufrj.br

Felipe Assis
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, RJ, Brazil
fassis@cos.ufrj.br

Patrícia Gonçalves
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, RJ, Brazil
patriciaamaralgurgel@ufrj.br

André Gheventer
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, RJ, Brazil
gheventer@ufrj.br

Larissa Galeno
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, RJ, Brazil
galeno@cos.ufrj.br

Arthur Sasse
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, RJ, Brazil
artsasse@cos.ufrj.br

Júlio César Guimarães
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, RJ, Brazil
jcguimaraes@cos.ufrj.br

Carlos Brito
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, RJ, Brazil
carloshenriquefbf@poli.ufrj.br

João Pedro Wieland
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, RJ, Brazil
jpvbwieland@cos.ufrj.br

ABSTRACT

The advances and availability of technologies involving Generative Artificial Intelligence (AI) are evolving clearly and explicitly, driving immediate changes in various work activities. Software Engineering (SE) is no exception and stands to benefit from these new technologies, enhancing productivity and quality in its software development processes. However, although the use of Generative AI in SE practices is still in its early stages, considering the lack of conclusive results from ongoing research and the limited technological maturity, we have chosen to incorporate these technologies in the development of a web-based software system to be used in clinical trials by a thoracic diseases research group at our university. For this reason, we decided to share this experience report documenting our development team's learning journey in using Generative AI during the software development process. Project management, requirements specification, design, development, and quality assurance activities form the scope of observation. Although we do not yet have definitive technological evidence to evolve our development process significantly, the results obtained and the suggestions shared here represent valuable insights for software organizations seeking to innovate their development practices to achieve software quality with generative AI.

KEYWORDS

Generative Artificial Intelligence, Software Engineering, Quality, Development.

1 Introduction

The software development cycle comprises structured phases that guide the conception, implementation, and maintenance of software systems [7]. Traditionally, this process has relied heavily on manual activities, making it prone to errors and frequent, unforeseen delays. As the demand for more agile and scalable solutions grows, it becomes imperative to incorporate technologies that can accelerate various phases of the development cycle and ensure software quality [12].

Advances in generative Artificial Intelligence (AI) technologies have driven significant transformations across various work processes, including Software Engineering (SE) [4, 10]. Technologies based on large language models (LLMs) and generative models can support the (semi-) automation of activities throughout the software development cycle. This movement is reshaping expectations regarding how software systems can be conceived, built, and maintained, to improve quality and productivity, reduce costs, and ensure faster delivery [14]. Despite the rapid progress in promoting these technologies, questions remain concerning their availability, effectiveness, limitations, and best practices when applied to real-world development projects [3].

In light of this transformative scenario, this experience report describes the development of a web-based software system in healthcare. The system was designed to support healthcare professionals in monitoring patients undergoing smoking cessation treatment. Our development team undertook this initiative in response to a request from the Institute of Thoracic Diseases (Portuguese:

Instituto de Doenças do Tórax, IDT) at the Federal University of Rio de Janeiro (UFRJ). We chose to utilize generative AI tools to support the development of the product. Beyond the effective delivery of a high-quality software solution, our interest lies in observing, in practice, how emerging technologies, including generative AI capabilities, can be integrated into the process of building quality software systems in real-world contexts.

As part of the software construction process, we mapped generative AI–based tools that could support different phases of the software development cycle [13]. The results revealed only a limited number of industrial solutions aimed at requirements elicitation, design, project management, coding, testing, experimentation, and, more broadly, quality assurance. However, we observed that empirical evidence regarding the effectiveness of these technologies in real-world projects remains scarce, despite numerous claims about their feasibility and capabilities. We view this scarcity of evidence as an opportunity for learning. To address this issue, we adapted our iterative and incremental development process to integrate generative AI–based tools as extensively as possible throughout the development cycle. Our observation was guided by the following research question: *"Is it possible to develop a software system using only generative AI–based tools across the different phases of the development cycle?"*

The results indicated that, although the exclusive application of generative AI throughout the development cycle does not yet enable the construction of a fully functional web system, its adoption significantly contributed to the advancement of development processes. The experience revealed progress, but also challenges, particularly in adapting traditional engineering practices to the new dynamics introduced by these technologies. One of the key findings was the emergence of new project artifacts, such as prompts, which began to demand attention comparable to that traditionally devoted to classical elements like requirements and test cases.

A prompt is an instruction or information provided to a generative AI tool to produce a response, perform a task, or guide the model's behavior [18]. The more precise, more specific, and more detailed the prompt is, the greater the likelihood of obtaining a practical, creative, and accurate response—applicable in diverse contexts such as text, image, and code generation, as well as translations, among others [18]. The prompt has become a central element in the communication between developers and generative AI technologies. The quality of the responses proved to be intensely dependent on the clarity, completeness, and intentionality expressed in the formulation of these prompts, which required developers to cultivate skills related to technical writing and the precise definition of objectives.

Another noteworthy element was Markdown, a lightweight markup language commonly used to format text quickly and efficiently, especially in digital environments such as blogs, forums, technical documentation, and development platforms like GitHub. With its simple syntax, based on familiar characters (such as asterisks, hashes, and brackets), Markdown enables the creation of headings, lists, bold and italic text, links, images, and code blocks without requiring complex commands. Its main advantage lies in clarity and practicality: the content remains readable even without rendering, facilitating editing and collaboration among developers [16].

Thus, throughout the project, Markdown proved essential for organizing the outputs generated by the models and documenting prompts, design decisions, and technical instructions. These artifacts came to play a significant role in quality assurance activities, particularly in requirements and testing, which required greater effort in inspecting and validating artifacts produced with the support of generative AI. This experience report describes the actions and observations carried out during the development process, highlighting the lessons learned regarding the strengths and weaknesses observed.

The remainder of this paper is organized as follows: section 2 presents the observation case; section 3 describes the development process with the use of generative AI; section 4 outlines the lessons learned; section 5 discusses the limitations of the reported experience; section 6 presents related works; and finally, section 7 presents the concluding remarks and directions for future work.

2 Observation Case: Web-Based Healthcare System

The web-based software system was conceived to replace the manual process of recording clinical information, which had previously been carried out through physical documents for notes on consultations and the progress of smoking cessation treatment at IDT/UFRJ.

The mission of IDT/UFRJ is to promote the continuous improvement of healthcare systems. Among its various areas of activity, its work on smoking cessation stands out, addressing both conventional cigarettes and vaping, with a focus on monitoring patients who seek to improve their quality of life by quitting smoking. To this end, patients undergo periodic clinical follow-ups, during which individual markers are collected and anamnesis is performed, to update medical records (maintained in separate files) and obtain data that enable the observation of treatment progression.

It has been observed that treatment success is closely linked to the frequency of patients' visits to the clinic and the motivations provided by the medical team. However, maintaining constant contact with each patient poses a challenge, which led IDT/UFRJ to decide on the development of a mobile application for Android, designed to support smokers in permanently quitting cigarettes. This application is currently being conventionally developed within our organization and enables the collection of relevant information for individual clinical monitoring.

Nevertheless, the information collected through the application must be transmitted and stored in a database located at IDT/UFRJ, where the healthcare team can access and use it. At that time, however, no web-based software solution with an integrated database existed to manage the reception, storage, and consultation of such data through dashboards and other functionalities to support clinical decision-making. This gap motivated the proposal to develop the web-based version of the system. A system of this kind, focused on monitoring smoking patients in the context of pulmonology, proves extremely valuable for organizing consultations, tracking clinical progress, and enabling personalized interventions, particularly for those under follow-up at IDT/UFRJ.

In this context, and considering that the Android application was under development and scheduled for delivery at the beginning of

the second semester of 2025, IDT/UFRJ envisioned, by mid-April of that year, the possibility of also developing the web-based component of the system. This new stage emerged from the need to address usage scenarios that had not yet been identified, designed, or implemented. The main challenge, however, is that IDT/UFRJ, like any public organization, faces constraints of time and resources: its healthcare teams are overburdened and require access to a functional software solution as soon as the mobile application becomes available. In light of this, our challenge was to identify engineering approaches and instruments that would enable the construction of an initial, operational, and high-quality version of the web application, even in the face of the limited continuous availability of the stakeholders involved.

The project involved two physicians from IDT/UFRJ, specialists in the problem domain who acted as the primary external stakeholders, and fifteen software engineers considered internal stakeholders, all with at least one year of industry experience (see Table 1). Among these engineers, two are not directly engaged in research. The team's educational background comprises one senior software engineer and researcher (project leader), five doctoral candidates, seven master's students, and two undergraduate students in their final year of study in Computing and Information Engineering. The software engineers were familiar with generative AI tools and traditional software lifecycle activities. However, until this case, none of them had used generative AI tools in an integrated manner across the entire software development process.

Tabela 1: Basic Profile of the Development Team

Developer	Area of Expertise	Professional Level
Dev1	Software Engineering	Master (Leader)
Dev2	Requirements Engineering	Mid-level
Dev3	Requirements Engineering	Mid-level
Dev4	Requirements Specification	Mid-level
Dev5	Requirements Specification	Mid-level
Dev6	Design	Mid-level
Dev7	Design	Mid-level
Dev8	Project Management	Senior
Dev9	Project Management	Junior
Dev10	Coding	Senior
Dev11	Coding	Mid-level
Dev12	Software Quality	Senior
Dev13	Software Quality	Senior
Dev14	Software Quality	Mid-level
Dev15	Design	Junior

Before the start of practical activities, participants received an introductory training on concepts related to the application of generative AI in Software Engineering. This step was necessary considering that most software development teams do not receive formal training on using LLMs and generally possess only basic knowledge of them [6]. After the training, the developers were organized into different teams according to their familiarity with the development phases. They were also introduced to the clinical context of the project, the current workflows for treating patients undergoing smoking cessation, and the challenges associated with manual record keeping.

The web-based software solution must address three user profiles: (i) healthcare professionals, responsible for registering, viewing, and updating patients' clinical and behavioral data, as well as recording observations during each visit; (ii) residents, with restricted access limited to viewing patient data; and (iii) administrators, with complete system access, including user and permission management. The system must also provide for integration with the mobile application used by patients themselves, through which behavioral data relevant to treatment are collected, such as relapses, withdrawal symptoms, and smoking history.

3 The Constructive Process with Generative AI

To examine the influence of generative AI technologies, we structured the development process around the classical stages of software engineering: requirements elicitation, use-case scenario collection, requirements specification, design, coding, testing, and management, as illustrated in Figure 1. Although presented sequentially, these stages unfolded iteratively as an investigative strategy rather than a waterfall model. This decision stemmed from the need to systematically observe the role of generative AI in each activity, ensuring clarity and traceability across artifacts.

To support this process, as mentioned earlier, the development team was organized into seven work groups: requirements elicitation (Dev1, Dev2, Dev3, and stakeholders), use-case scenarios (Dev1 and stakeholders), requirements specification (Dev4 and Dev5), software design (Dev6, Dev7, and Dev15), project management (Dev8 and Dev9), coding (Dev10 and Dev11), and verification, validation, and testing (VV&T) (Dev12, Dev13, and Dev14).

Each group was responsible for carrying out the activities of its respective phase, which included creating prompts, interacting with generative AI tools, and producing the corresponding artifacts. The project management and VV&T teams worked transversally across the other activities. Developers from different groups evaluated and validated these artifacts, as illustrated in Figure **??**. Furthermore, Table 2 provides an overview of the generative AI tools and the input data (model inputs) and outputs of each phase.

To interact with the AI tools, the prompts were structured according to Google's[1] guidelines for the use of generative AI tools. These prompts were organized into five main components: (i) **persona**, which defines the model's role as a software requirements expert; (ii) **task**, which describes the activity to be performed, such as the specification, classification, and prioritization of requirements; (iii) **context**, which provides a description of the system and the documents produced during the initial phases; (iv) **example**, presenting samples of functional and non-functional requirements; and (v) **output format**, which specifies how the generated content should be structured. It is essential to note that each artifact produced in the project had to be converted into Markdown (.md) format to be input into the AI models. Table 3 presents an excerpt of the prompt used in the requirements specification stage. The following subsections detail each phase of the process and the tools employed.

Additionally, the artifacts produced and validated throughout the project are available[2], including: (i) the structured prompts used for interactions with generative AI tools; (ii) the documents employed as inputs to the generative AI tools; and (iii) the artifacts generated by these tools. It should be noted, however, that some documents, such as the clinical history form used at IDT/UFRJ and the requirements document, were not made available because they

[1]Prompting Guide 101: https://services.google.com/fh/files/misc/gemini-for-google-workspace-prompting-guide-101.pdf

[2]Artifacts available at: https://doi.org/10.5281/zenodo.16366063

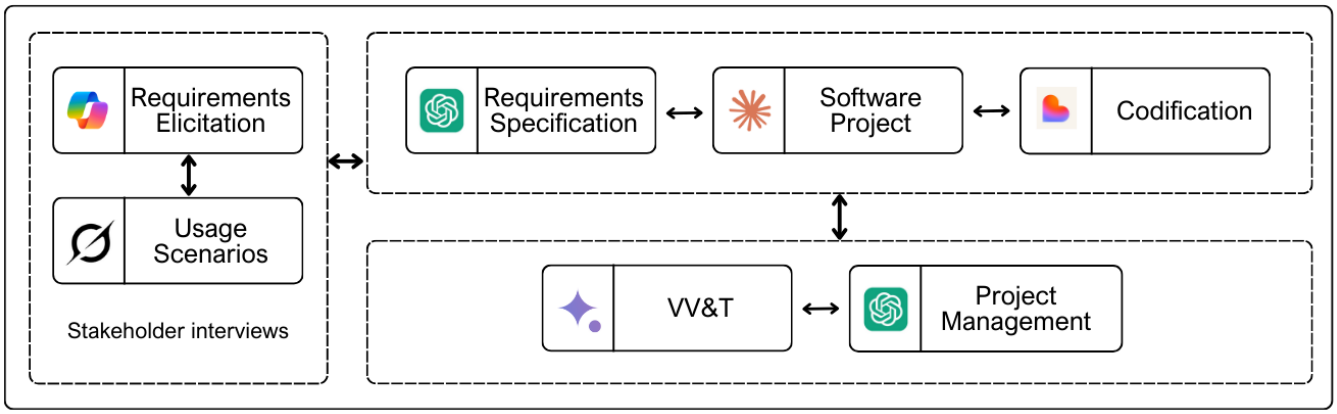

Figura 1: Constructive Process with Generative AI

Tabela 2: Artifacts Generated Throughout the Constructive Process with AI

Phase	Input	Tool or Model Used	Outputs	Role of AI
Requirements Elicitation	Stakeholder interviews	MS CoPilot (https://copilot.microsoft.com/)	Vision document	To enhance textual clarity and articulation.
Use Cases	Requirements document, clinical history form	ChatGPT-4.1 (https://chatgpt.com/)	Use Case document with main, alternative, and exception flows	To generate initial flows (main, alternative, exception).
Requirements Specification	Vision document, use-case scenarios, clinical history form	ChatGPT-4o (https://chatgpt.com/)	Document with 41 requirements, including functional (28) and non-functional (13)	To generate the initial set of requirements, as well as classification and prioritization.
Design	Requirements document, clinical history form	Claude-3.5 (https://claude.ai/)	Architecture diagram, Data model	To support the definition of the system architecture, components, and data flows.
Project Management	Vision document and use-case scenarios	ChatGPT-o4-mini (https://chatgpt.com/)	Project Plan (classical model)	To elaborate a project plan, considering timelines and general intentions regarding the use of generative AI tools.
Use-Case Scenarios	Vision document	Grok (4.0) (https://grok.com/)	Document with eight use-case scenarios	To generate use-case scenarios.
Coding	Requirements document, clinical history form, use cases	Lovable https://lovable.dev	Backend, frontend	To generate frontend and backend code.
Verification, Validation, and Testing	Requirements document, clinical history form, use cases	Gemini 2.5 Flash https://gemini.google.com/	Document with Test Plan and test case suggestions	To generate test plans and mappings between use cases and requirements.

contained exclusive and sensitive information from the developed software project.

3.1 Requirements Elicitation and Use-Case Scenarios

The process began with the requirements elicitation phase, which aimed to understand the system's needs, expectations, and constraints. Developer Dev1 interviewed two medical specialists and mapped the current clinical management workflows to identify problems, limitations, and potential areas for improvement.

Based on the interviews, the initial version of the vision document was drafted. Dev1 used Copilot (free version) on Windows 11 to refine the text, benefiting from its ease of access and capacity to enhance clarity and textual articulation [15]. The document was then submitted to Grok (free version), which generated seven use-case scenarios based on prompts and the document's content. Dev1 integrated these scenarios into the vision document and submitted it for review by stakeholders (Dev3 and Dev4). They approved all scenarios and suggested including two additional ones, specific to IDT/UFRJ, as well as the patient's clinical history form. They also recommended stylistic adjustments, which were incorporated. Following a subsequent review by Dev2 and Dev3, which confirmed consistency with the domain, the document was presented to the team and became the guiding reference for the subsequent project stages. The process lasted about two weeks and was praised by external stakeholders for its clarity and the agility with which the problem was described.

3.2 Requirements Specification

The requirements specification was carried out with the support of the ChatGPT model (free version), chosen for its ease of access, the team's familiarity with the tool, and because previous studies have highlighted its potential to transform software requirements [9][2]. The tool helped define, classify, and prioritize requirements. The primary sources of information were the vision document, the use-case scenarios, and a clinical history form used by physicians to record information about patients undergoing treatment, as illustrated in Figure 2.

Figure 2 illustrates an example of generating requirements and use cases. This process occurred iteratively, resulting in nine versions of the requirements document. Each version was inspected by developers Dev1, Dev2, and Dev3 through ad-hoc inspections that identified omissions, redundancies, and ambiguities. The issues raised guided new interactions with the generative AI model, enabling progressive refinement of the content. The final version was reviewed by six developers (Dev1, Dev2, Dev3, Dev10, Dev14, and

Tabela 3: Excerpt of Prompt Used in Requirements Specification

Persona
You are a senior software requirements engineer.
Task
You have been assigned to construct functional and non-functional software requirements based on the context provided below. You must also prioritize the requirements within the categories [essential, important, desirable].
Context
The Institute of Thoracic Diseases at the Federal University of Rio de Janeiro (UFRJ) has the mission of continuously improving healthcare systems. Among its many research and knowledge advancement fronts, the anti-smoking initiative (covering both conventional cigarettes and VAPE) stands out, focusing on monitoring smoking patients who seek to improve their quality of life by quitting. To this end, patients undergo periodic clinical follow-ups, during which individual markers are collected and anamnesis is performed, with the aim of updating their medical records (stored in individual files) and obtaining data that allow monitoring the progress of treatment. This information needs to be transmitted and stored in a database located at IDT/UFRJ, where it can be accessed and used. The proposal document presents different scenarios. The focus at this moment is on scenario 1: **1. Patient Registration and Profile** - A new smoking patient arrives at the clinic. The pulmonologist needs to record detailed information for follow-up. **System use:** - Registration of personal data (name, age, contact information, etc.). - Recording smoking history: number of cigarettes per day, years of smoking, calculation of pack-years. However, other scenarios must also be considered, taking into account the system's evolution and dependencies between requirements. In summary, the goal of the system is to be a web-based solution that allows healthcare professionals to register and monitor their patients. In addition, the system must enable healthcare professionals to monitor patients who are registered in the associated mobile application.
Examples
Below are some preliminary examples of how functional requirements should be described: - The system must register a new patient through a form, as described in the *Markdown* file. - The system must allow the registration of visits for already registered patients. - The system must present a list of registered patients, displaying their name and status. Below are some preliminary examples of how non-functional requirements should be described: - The system must be adaptable to different screen sizes. - The system must be compatible with browsers available on the market.
Output Format
I want you to generate one table for functional requirements (FR) and another for non-functional requirements (NFR) in the following format: Requirement ID | Description | Priority FR 01 | The system must... | Essential

Dev15), who suggested minor adjustments that were incorporated before the document was finalized.

With the requirements consolidated, use cases were developed, covering primary, alternative, and exception flows. Initially, the ChatGPT model generated one use case per requirement, resulting in fragmented artifacts. An example was included in the prompt to address this problem, which resulted in more cohesive, functional, and domain-aligned use cases. The process lasted about two weeks and resulted in seven versions of the use-case document.

3.3 Software Design

The software design phase defined the system architecture, components, and data flows. It was carried out by Dev6 and Dev7, with support from the Claude Opus 4 model (free version). The prompt was based on the requirements document and the clinical history form. The Claude.AI model was chosen for its technical capability to process text and images simultaneously, making it suitable for tasks involving visual schematics and detailed technical descriptions.

The first version of the design models presented shortcomings, including redundancies and ambiguities. After a manual review, a second, more cohesive version was produced through a new interaction with the AI. This version underwent a collaborative review with Dev1, Dev10, and Dev11, who identified additional inconsistencies. Following a third refinement of the prompt, a consolidated design model was obtained, aligned with the system's overall objectives. The entire process lasted about one week.

3.4 Project Management

This phase aimed to establish the development processes that should be followed. The first project management model was created based on the vision document and the initial use-case scenarios. To this end, the project management team used ChatGPT to generate the project plan. A prompt was prepared, and the timelines and the general intentions for using generative AI technologies were explicitly provided. However, the generated plan did not meet the project expectations, as it presented activities aligned with conventional development models, such as MPS.BR[3], without integrating these new technologies into project activities.

A second version of the project plan was generated after the first version of the system had undergone a complete transformation cycle. The suggested plan adhered to the initial outline, proposing

[3]General Guide to MPS for Software: 2024: https://softex.br/download/guia-geral-mps-de-software2024/

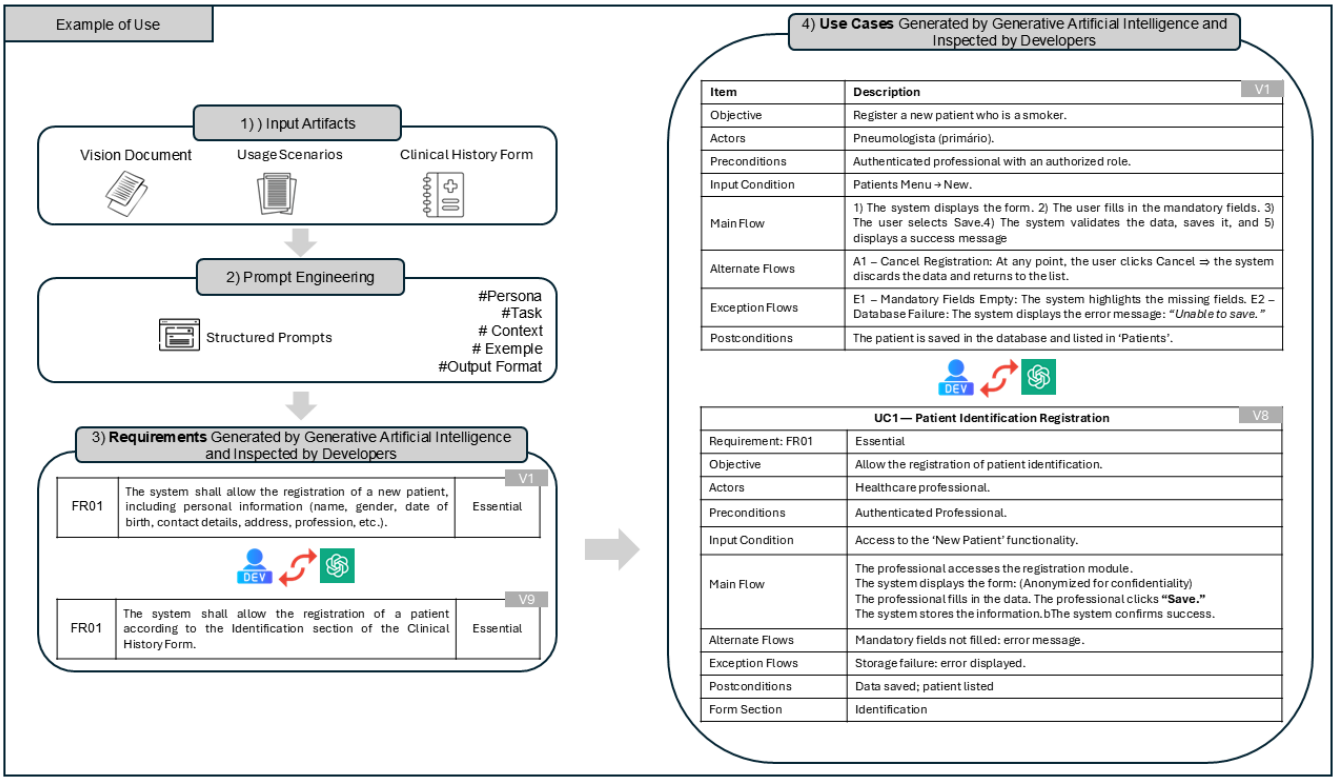

Figura 2: Example of the Requirements Specification and Use Case Process

activities and tasks that were incompatible with the system's development constraints. Had this plan been followed strictly, not only would it have delayed deliveries, but it might also have compromised the effective construction of the design models, hindering the continuous adaptation required in projects involving the use of generative AI. This experience demonstrated that, although generative AI tools can assist in drafting initial plans, they still present significant limitations in adapting to specific and dynamic contexts, requiring critical supervision and manual adjustments by the project team.

3.5 Coding

The coding phase implemented the requirements, use cases, and design into a functional web system, encompassing the development of the frontend, backend, and integration layers.

The choice of the supporting tool was based on a prior mapping of generative AI solutions, including v0, Bolt[4], Replit[5], and Lovable. All were evaluated using the same prompt and the clinical history form. Lovable was selected because it best met the defined requirements and integrated the different layers of the application more effectively.

Lovable is a generative AI–based development tool that enables the creation of web applications through natural language commands. It automatically generates frontend and backend code, accepts text and images as input, and provides real-time visual editing. Because it is intuitive and efficient, it suits developers, designers, and non-technical users seeking to create digital solutions quickly. However, its use requires a paid subscription. Although the project initially intended to rely solely on open-access technologies, we opted for Lovable due to the availability of credits provided to the developers.

Using the tool required the creation of design artifacts, such as a knowledge base containing a system description, objectives, user profiles, functional and non-functional requirements (including security, performance, and responsiveness), and visual guidelines (color palette and typography defined through sample images). The defined stack included a frontend built with React and Tailwind CSS, and a Supabase (PostgreSQL) backend. Each specified use case was then transformed into an individual prompt, structured according to best practices recommended by Lovable[6], known as the CLEAR framework (Concise, Logical, Explicit, Adaptive, and Reflective) [8].

The implementation followed Lovable's recommendations, starting with essential functionalities such as authentication and user management. After review, additional functions were developed, including patient registration, smoking history, and automated calculations based on forms and access profiles. The tool generated proposals aligned with the requirements but required fine-tuning through multiple iterations using the *"Try to fix"* feature. Short

[4]Bolt: https://bolt.new/
[5]Replit: https://replit.com/
[6]Lovable - Prompt Engineering: https://docs.lovable.dev/prompting/prompting-one

new prompts were employed to correct display issues or adjust the interface.

3.6 Verification, Validation, and Testing

The verification, validation, and testing (VV&T) team initially selected ChatGPT, considering its popularity, accessibility, and the preliminary results of the AI technology mapping previously conducted. The first version of the test plan was generated based on the vision document, the use-case scenarios, and the patient's clinical history form. To guide the generation, the prompt adopted the persona of a "Senior Software Test Analyst".

The initial generation of test cases with ChatGPT was discontinued due to insufficient detail and the limitations of the free version. The team then adopted Gemini as the primary tool. The test plans remained under-detailed despite reusing the prompt, necessitating adjustments through the use of new prompts. As the project advanced, it became necessary to integrate updated versions of the documents. However, the model struggled to assimilate the information, requiring it to be resent.

Gemini inserted citation markers during generation, which were later removed at the team's request. The team also requested the inclusion of a menu with links, greater detail on exception cases, and corrections to user profiles, specifically replacing the "ADMIN" profile with "Healthcare Professional" and "Resident". Finally, the test plan was refined to ensure all functionalities were covered. In addition, a template for the test plan was created. The generation of the second version required refactoring the input prompt and expanding and aligning the instructions with the requirements team's prompting guidelines, which ensured greater cohesion in the project.

During the VV&T phase, the test plan identified associations between use cases and requirements not specified in the original documentation. The requirements team reviewed these associations and adjusted the test plan accordingly. After several iterations and refinements of the prompts, the final plan was approved by the VV&T team, meeting the project's objectives and quality standards. Automated test generation followed an exploratory approach using ChatGPT and Gemini. Other models, such as DeepSeek[7] and Claude, were employed with standardized prompts for infrastructure and interfaces. While ChatGPT and DeepSeek showed persistent shortcomings, Claude produced functional tests with minimal human intervention. The effectiveness of these tools depended directly on the quality of the prompts and specialized oversight. The experience demonstrated that generative AI tools, although valuable for automating and generating test artifacts, still require continuous adjustments and careful human validation.

4 Lessons Learned

The experience of using generative AI, specifically the free version of ChatGPT, for the **requirements specification** of the web system demonstrated that full automation of these activities is not yet feasible. Although the tool provided promising initial support, four rounds of manual reviews and adjustments by the developers were required to achieve an acceptable outcome in terms of quality and coherence. During this process, the requirements document generated by the model exhibited several defects, such as the omission of critical requirements, duplication of similar items, and inadequacies in requirements prioritization, which compromised the clarity and usefulness of the document in guiding subsequent development phases. These issues demanded iterative inspection and refinement to correct inconsistencies and fill gaps.

Concerning use case generation, it became evident that a more sophisticated and detailed example needed to be included in the prompt to ensure that the model established a coherent and integrated association between requirements and use cases. This prevented the generation from resulting in mere one-to-one correspondences that failed to reflect the system's real interdependencies and flows. Additionally, a technical limitation was observed in generating use cases, which often appeared incomplete. In most cases, at least two additional commands had to be submitted to obtain a complete set of use cases. A noteworthy aspect identified was the higher incidence of errors and "hallucinations", incorrect or unfounded responses, in the later parts of the generated outputs. This reinforced the need to "remind" the language model of the whole context in each request, ensuring it had access to the complete history and all necessary information to produce consistent and coherent outputs.

An additional challenge concerning the "memory" and interaction history with ChatGPT was identified during the development process. At the beginning of the requirements generation, the model produced an incorrect specification of a functionality, which was manually corrected in the document's final version. However, the same functionality was described with the same inaccuracy during the generation of the use cases. This episode highlights the difficulty of ensuring the correctness of generative AI outputs, even after detailed manual inspections of the artifacts.

Thus, generative AI for requirements specification can be highly advantageous, as it facilitates and accelerates the process. Nevertheless, constant human supervision remains indispensable, with verification and validation of the generated requirements essential to ensure their quality. In this study, the application of ad-hoc software inspections proved fundamental to ensuring the reliability of the produced artifacts. Even so, context management by the model and constructing optimized prompts remain significant challenges for generating an accurate and consistent specification.

The use of generative AI in **software design** demonstrated potential to accelerate the initial creation of architectures, diagrams, and data models, although it also revealed essential limitations. One of the main lessons learned was that the level of detail and clarity in the prompts directly influences the quality of the generated artifacts. Generic or vague prompts, even when grounded in design documents, led to simplified architectures, overlapping responsibilities, unclear data flows, and the omission of essential information. To mitigate these deficiencies, it was necessary to provide detailed descriptions of components, interaction flows, and technical constraints and include explicit examples of validated diagrams and similar models when crafting the prompts.

Another challenge was maintaining consistency across multiple iterations, which were not always incremental in nature. With each new generation of artifacts or adjustments, the generative AI tool often disregards previously established details or alters already validated design parts. This required strict context management,

[7] DeepSeek: https://www.deepseek.com/

including detailed tracking of changes, constant supervision, and maintaining parallel documentation to version the obtained results. The experience demonstrated that generative AI has the potential to serve as a valuable tool to support software design, contributing to both productivity and creative inspiration. However, its use requires extremely well-structured prompts, clear examples, and the continuous and critical involvement of human architects. Once again, inspection was fundamental to ensuring the quality of the generated artifacts. Notably, the design models proved helpful for team understanding and envisioning the potential software solution. Nevertheless, these models had little influence on coding, suggesting the need to assess their necessity in similar design cases.

In **project management**, we observed that the tool reproduced a traditional development model misaligned with the project's real demands, particularly given the constraints and dynamics of a public-sector and innovation-driven context involving AI technologies. This demonstrated that simply automating outdated processes is insufficient: it is essential to adapt management practices to reflect the particularities of applying new technologies, such as those based on generative AI, while considering the need for flexibility, short iterative cycles, and reduced dependence on stakeholder presence. Otherwise, planning may become a barrier and hinder the project's progress.

In the **coding** phase, we identified an intrinsic relationship between the quality of the outcomes obtained in the earlier requirements and use-case stages and the level of detail provided to the Lovable tool. Generic or poorly structured prompts often led to incomplete functionalities or unexpected behaviors. In this context, the importance of using documents in Markdown (.md) format was reinforced, as it allowed for a clear hierarchical structure of information, enabling precise descriptions of tasks and use cases, as well as the explicit detailing of fields, data types, and their mandatory status, where applicable.

Including images as references in the prompts was essential to ensure visual consistency across the interfaces, thereby reducing unexpected changes throughout iterations. Dividing development by use cases also proved effective for modularizing the process. However, more complex functionalities—such as automated scoring, permission control based on user profile, and dynamic interface updates—were not correctly implemented within a single iteration. These limitations highlighted the tool's difficulty in interpreting complex logical rules, maintaining state across components, and ensuring consistency over time. Multiple interactions and targeted adjustments were required to achieve the expected results.

It was further observed that, as the system evolved, applying specific changes via prompt became increasingly complex, as unrequested interventions began to occur more frequently. Thus, we conclude that the use of generative AI in software system development requires a clearly defined scope, modularity in development, the creation of supplementary artifacts, and particular attention to the tool's limitations when dealing with complex business rules and interactive interfaces.

To address such limitations and enhance the efficiency of the development process with generative AI technologies, we suggest integrating the Lovable tool (or another similar agent-based platform) with complementary tools. We observed that a viable approach is to combine the initial application generation provided by the tool with targeted corrections and more precise adjustments carried out through general-purpose LLMs such as ChatGPT. It is also advisable to consider exporting the generated code to advanced editors, such as Cursor[8], with direct integration into GitHub[9], enabling generative AI-assisted editing, more robust version control, and more effective collaboration in problem resolution. It is worth noting that such an approach requires greater developer expertise, as they must identify which code components need adjustments before submitting them to the assistant or requesting a specific correction.

The experience with generative AI tools such as ChatGPT and Gemini in **quality assurance** activities highlighted the potential of these technologies to support specific SE tasks. In particular, they proved helpful in the initial generation of artifacts and for accelerating repetitive or operational tasks, such as structuring test scenarios or identifying basic cases from simple functional requirements. However, practical experimentation also revealed essential limitations. The generated test plans lacked depth in many cases, failing to cover edge cases, exceptions, or domain-specific aspects of the application. In addition, it was common for the outputs to include generic information or content disconnected from the project's real context, indicating that generative AI struggles to correlate multiple technical sources (such as requirements documents, business rules, and system specifications) and correctly interpret more complex nuances.

For example, the initial deployment of the web system was also carried out through Lovable, which facilitated process monitoring by providing automatic cloud hosting and a public URL for application access. A noteworthy feature was the platform's native integration with Supabase, an open-source solution responsible for authentication, data storage in PostgreSQL, and enforcing Row Level Security (RLS) policies. This integration is performed via an API Key, allowing Lovable to execute operations in the database. However, it was observed that the code generated by Lovable embedded the API key directly into the source code, an inadequate practice that may expose sensitive information and compromise security. This flaw was manually corrected by transferring the credentials to a .env file and correctly importing them into the runtime environment.

Additionally, the precision of the prompts proved crucial for configuring access rules to ensure the application's security and consistency. Poorly defined prompts may lead the LLM to suggest removing authentication restrictions or weakening RLS policies, creating risks to data integrity. This type of vulnerability is documented in the Common Vulnerabilities and Exposures[10] (CVE-2025-48757) of the tool, which details this type of exposure. Finally, to enable application execution in local environments, an additional prompt had to be issued to Lovable, requesting the generation of configuration files in Docker[11].

These limitations required additional inspection efforts from the team, particularly in validating acceptance criteria, ensuring the completeness of scenarios, and verifying adherence to testing

[8]Cursor: https://cursor.com/
[9]GitHub: https://github.com/
[10]Common Vulnerabilities and Exposures: https://nvd.nist.gov/vuln/detail/CVE-2025-48757
[11]Docker: https://www.docker.com/

objectives. This process revealed that, although AI is promising, it still relies heavily on the critical role of qualified professionals, who must review, complement, and adapt the generated results to ensure the quality of the software produced. Another key observation was the importance of prompt engineering. As mentioned earlier, the more specific, contextualized, and well-structured the prompt, the greater the likelihood of obtaining functional responses aligned with the project's needs. This clarifies that technical expertise is desirable and essential to transforming generative AI into a faithful ally in quality processes.

Thus, generative AI in quality assurance activities should be regarded as support rather than a professional replacement. The effectiveness of these tools depends directly on prompt engineering, a still-emerging area with limited guidance, and the critical involvement of experienced professionals, reinforcing the importance of manual reviews and inspections as cornerstones of software quality. Similar to what was observed in the companies studied, we also noted that compliance with legal requirements was often considered sufficient, highlighting the need to treat ethics as a non-functional requirement and actively integrate it into development practices.

This experience demonstrated that ethical issues cannot be addressed in isolation or superficially; they require a comprehensive approach. Responsibility and transparency were fundamental to ensuring that the generated artifacts were critically evaluated, corrected, and traceable. At every stage—from requirements elicitation to validation, verification, and testing—it was necessary to maintain active human oversight to ensure that the AI outputs aligned with the objectives of the web system. In line with the findings of Baldassarre *et al.* [1], we learned that ethics must be integrated into the development process from the very beginning. Moreover, considerations of fairness also proved relevant. When elaborating requirements and use cases with the support of generative AI, it was necessary to ensure that no biases or omissions were introduced that could disadvantage specific user profiles.

5 Limitations of the Reported Experience

Despite our efforts to ensure rigor in the conduct and documentation of this experience report, we acknowledge certain limitations that may influence the interpretation of the results.

For instance, evaluating the quality of artifacts generated by generative AI tools involved a degree of subjectivity on the part of the developers. To mitigate this aspect, we adopted strategies such as peer reviews, internal discussions, and the documentation of artifact traceability, aiming to enhance consistency and transparency in the evaluations. Moreover, participants' prior knowledge of generative AI and their level of engagement in the activities may have directly influenced the observed results, thereby limiting the depth of the analyses and the generalizability of the conclusions across different domains.

We also acknowledge that the observations were conducted within a single project, involving a specific team of developers and a particular development context, which may restrict the generalizability of the findings. To mitigate this limitation, we provided a detailed description of the investigated setting, the tools employed, and the criteria guiding decision-making, enabling other professionals to assess the applicability of the results in their own contexts. Moreover, the division of the development team and the choice of AI tools, although justifiable within the context of the study, may have restricted the scope of the answers to the research question and the generalizability of the results.

Finally, the rapid evolution of generative AI tools presents an additional challenge, as some observations may quickly become outdated. To address this issue, we explicitly documented the versions of the tools and the time periods during which they were applied, thereby ensuring clarity regarding the technological context in which the experiences were conducted. It is also worth noting that the tools were employed in their free versions, which imposed technical restrictions (such as context limits, tokens, and functionalities) that may have influenced the results.

Beyond these limitations, it is essential to consider the ethical implications associated with using generative AI tools. The outputs produced by these systems may reflect biases present in their training data, raise questions about the intellectual authorship of the generated artifacts, and influence decisions without providing adequate explainability. Although this report did not identify critical cases, the absence of systematic mechanisms (e.g., ethical checklists, auditing tools) to evaluate such implications may constitute a limitation. Future investigations could explore these aspects in greater depth, incorporating specific ethical guidelines and mitigation strategies to foster the responsible use of AI.

6 Related Work

Recent studies have investigated the application of generative AI-based tools to facilitate specific tasks in software development. Zhao *et al.* [17] introduced ReqGen, a keyword-based approach for automatically generating software requirements using the UniLM model. Complementarily, Nair and Thushara [11] developed NL2Code. This hybrid framework combines Natural Language Processing (NLP) and Model-Driven Engineering (MDE) to translate natural language requirements and UML diagrams into executable code.

Despite these advances, such studies primarily focus on the use of generative AI in isolated tasks of the development lifecycle and on experiences related to the productivity of individual developers [5][3]. In contrast, the present study adopts a broader perspective, applying generative AI tools across multiple activities: requirements elicitation and specification, design, project management, coding, testing, experimentation, and, transversally, quality assurance. In this way, our work advances beyond prior research by demonstrating how generative AI can be incorporated systematically and integratively into a software project, moving beyond support for isolated activities.

7 Conclusion

The experience reported here, conducted in a real-world context of developing a web-based software system for public healthcare, enabled us to reflect on the emerging role of generative AI tools in core activities of the software lifecycle, including project management, requirements engineering, interface design, construction, and quality assurance. In response to our guiding question, whether it would be possible to develop a quality software system using only generative AI–based tools throughout the entire development cycle,

we conclude that the answer remains negative. Despite productivity gains and the usefulness of these tools in specific tasks, the system's quality was only achieved thanks to continuous human involvement, particularly in crafting effective prompts, performing technical inspections, and validating generated requirements and artifacts. Furthermore, the use of these technologies introduced the need for new artifacts in the project (prompts, Markdown artifacts), which required the inclusion of additional activities to ensure software quality. Thus, the experience reinforces that generative AI can serve as an ally in the development process but does not yet replace the critical role of **human expertise** in ensuring software quality.

Among the key lessons learned, we highlight that the use of these tools can indeed yield productivity gains, especially in the early stages of artifact or prototype construction, even when accounting for the emergence of new artifacts and development activities. Generating test drafts and interfaces from structured textual descriptions proved useful in accelerating iteration cycles and fostering alignment between the technical team and stakeholders. However, the potential of these tools is strongly conditioned by the quality of human–AI interaction. Prompt engineering emerged as a critical competence, yielding the best results when prompts were derived from well-structured documents with precise details and unambiguous language. Coding and detailed descriptions in Markdown, for instance, proved to be an effective strategy for organizing information and guiding the generation of relevant artifacts. Nevertheless, practical application revealed significant limitations: the tools struggled to comprehend complex relationships between documents, interpret more sophisticated business rules, and maintain consistency across different parts of the system. In regulated contexts, such as healthcare, these gaps represent concrete risks and require that specialists carefully inspect all generated outputs.

In this sense, the experience underscores that: (i) generative AI should be regarded as a **complementary tool**, not a substitute; (ii) **well-structured documentation expressed in clear language** is an essential condition for effective interaction with generative systems; (iii) the role of **human review and technical inspection remains irreplaceable in the development cycle**, particularly when pursuing quality, compliance, and trust; and (iv) we must **critically assess our current models of software development processes**, considering the new artifacts and roles required to construct software products with AI effectively.

Finally, we recommend that teams interested in adopting these technologies in software engineering contexts do so incrementally, with systematic validation processes, and with the recognition that the actual value of generative AI emerges when combined with the expertise of professionals who understand the domain, the system's objectives, and the tools' limitations. Special attention should be given to the project plan. The classical structures of process organization typically presented in current maturity models need to evolve to account for integrating these new technologies. As future work, we intend to further integrate generative AI tools into the development cycle, with a particular focus on automation and human validation. We also aim to investigate best practices for prompt engineering, compare artifacts produced with and without AI, and assess the use of these technologies in larger teams with different profiles. We also intend to explore the same approach in agile processes to evaluate whether the results hold in this context. Furthermore, we plan to examine the role of generative AI in requirements engineering within our software projects, expanding the evidence of its practical application in software engineering.

ARTIFACT AVAILABILITY

The materials used in the project are available at: https://doi.org/10.5281/zenodo.16366063.

ACKNOWLEDGMENTS

The authors express their deep gratitude to the Instituto de Doenças do Tórax of the Federal University of Rio de Janeiro, represented by Dr. Michelle Cailleaux and Dr. Natália Blanco, for entrusting our team with the software development. The CBV and FBG agencies support this experience. We acknowledge using the Copilot and ChatGPT tools to support refining our original text and improving its clarity. We carefully reviewed all outputs to ensure alignment with our original intent. Professor Travassos is a Research Fellow of CNPq and CNE of FAPERJ. CAPES, FAPERJ, and CNPq partially fund this work.

REFERENCES

[1] Maria Teresa Baldassarre, Domenico Gigante, Marcos Kalinowski, and Azzurra Ragone. 2024. POLARIS: A framework to guide the development of Trustworthy AI systems. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI*. 200–210.

[2] Mario Binder and Vitaliy Mezhuyev. 2024. A framework for creating an IoT system specification with ChatGPT. *Internet of Things* 27 (2024), 101218.

[3] Mariana Coutinho, Lorena Marques, Anderson Santos, Marcio Dahia, Cesar França, and Ronnie de Souza Santos. 2024. The role of generative AI in software development productivity: A pilot case study. In *Proceedings of the 1st ACM International Conference on AI-Powered Software*. 131–138.

[4] Ágatha de Almeida, Eliane Collins, and Ana Carolina Oran. 2024. AI in Service of Software Quality: How ChatGPT and Personas Are Transforming Exploratory Testing. In *Proceedings of the XXIII Brazilian Symposium on Software Quality (SBQS '24)*. 179–188. doi:10.1145/3701625.3701657

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

[6] Fabiano Damasceno Sousa Falcão and Edna Dias Canedo. 2024. Investigating Software Development Teams Members' Perceptions of Data Privacy in the Use of Large Language Models (LLMs). In *Proceedings of the XXIII Brazilian Symposium on Software Quality*. 373–382.

[7] Ralf Kneuper. 2017. Sixty Years of Software Development Life Cycle Models. *IEEE Annals of the History of Computing* 39, 3 (2017), 41–54. doi:10.1109/MAHC.2017.3481346

[8] Leo S. Lo. 2023. The CLEAR path: A framework for enhancing information literacy through prompt engineering. *The Journal of Academic Librarianship* 49, 4 (2023), 102720. doi:10.1016/j.acalib.2023.102720

[9] Nuno Marques, Rodrigo Rocha Silva, and Jorge Bernardino. 2024. Using chatgpt in software requirements engineering: A comprehensive review. *Future Internet* 16, 6 (2024), 180.

[10] MDB Modi. 2024. Transforming software development through generative AI: a systematic analysis of automated development practices. *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol* 10, 6 (2024), 536–547. doi:10.32628/cseit24106197

[11] Reshma P Nair and MG Thushara. 2025. NL2Code: A Hybrid NLP and Model-Driven Framework for Automated Code Generation from Natural Language and UML. In *2025 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*. IEEE, 1–6.

[12] K R Raghi, K Sudha, Sreeram A M, and Steve Joshua S. 2024. Software Development Automation Using Generative AI. In *2024 International Conference on Emerging Research in Computational Science (ICERCS)*. 1–6. doi:10.1109/ICERCS63125.2024.10894980

[13] Sabrina Rocha, Rodrigo Feitosa, Larissa Galeno, and Guilherme Travassos. 2025. A Metaprotocol For a Family of Rapid Multivocal Reviews of Generative AI in the Software Industry. In *Proceedings of the XXXIX Brazilian Symposium on Software Engineering*. doi:10.5753/sbes.2025.11577

[14] Jaakko Sauvola, Sasu Tarkoma, Mika Klemettinen, Jukka Riekki, and David Doermann. 2024. Future of software development with generative AI. *Automated Software Engineering* 31, 1 (2024), 26.

[15] Cristina Vasilescu and Militaru Gheorghe. 2024. Improving the performance of corporate employees through the use of artificial intelligence: The case of copilot application. In *Proceedings of the International Conference on Business Excellence*, Vol. 18. Sciendo, 1819–1830.

[16] Yihui Xie, Joseph J Allaire, and Garrett Grolemund. 2018. *R markdown: The definitive guide*. Chapman and Hall/CRC.

[17] Ziyan Zhao, Li Zhang, Xiaoli Lian, Xiaoyun Gao, Heyang Lv, and Lin Shi. 2023. Reqgen: Keywords-driven software requirements generation. *Mathematics* 11, 2 (2023), 332.

[18] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. In *The eleventh international conference on learning representations*.