Effective use of AI tools in a Software Engineering working environment. A comparison between Software Engineering and Business Administration students

Matei Avram
matei.avram01@gmail.com

July 1, 2025, 30 pages

Academic supervisor:	dr. Ana Oprescu, a.m.oprescu@uva.nl
Daily supervisor:	Bogdan Andrei, MSc, b.m.andrei@uva.nl
Host organisation/Research group:	UvA

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Abstract

There has been increasing uncertainty regarding the effect Large Language Models (LLMs) will have on the job market ever since the launch of ChatGPT on November 30, 2022. While the model is seemingly able to automate a significant portion of tasks in the Software Engineering industry, there have been concerns regarding fundamental issues of this technology (i.e. hallucinations).

We hypothesize these could be dealt with more effectively through increased domain knowledge and prompt engineering. As such, we aim to answer the question: ”To what extent can software engineering skills translate into effective AI usage performance compared to business students in a real life engineering scenario?”. We decided to compare Software Engineering (SE) and Business Administration (BA) students as they represent the two leading industries in terms of LLM adoption.

To answer the question we conducted an experiment of the two groups, where participants were asked to solve 3 tasks in a E-Ticket web application we developed, while having access to ChatGPT o4-mini reasoning model. We measured task success rate, project quality through SonarQube and we manually analyzed students' conversation to look for possible prompt engineering techniques used.

Our analysis suggest on average, SE students outperformed BA students in this experiment. We identified some quality issues introduced by BA students into the codebase as they were working extensively with the LLM sometimes without providing enough context. Finally, SE students were more proficient in creating prompts then BA students. However, prompt success rate should be attributed to the model's ability to understand user requests rather than the student's ability to convey them.

Thus, to operate and AI models effectively in a Software Engineering environment, domain expertise is required. Additionally, a prompt engineering course would help students convey their requests to LLMs more effectively.

Contents

Chapter 1

Introduction

With the launch of ChatGPT on November 30, 2022, AI has been a central topic in the scientific community as well as the general public. Products like ChatGPT, Gemini, DeepSeek and many more have cemented the belief that Large Language Models (LLMs) are a potential form of disruptive technology. As such, people in the job market have been looking at how it is possible to utilize it effectively.

Through marketing of the companies that developed these AI models, as well as research from the scientific community, one of the potential uses identified for this technology is automation in the Software Engineering industry [1]. As such, there is a widespread adoption of LLMs by software engineers, in order to help with more tedious issues like: debugging, prototyping, documentation, sometimes even fast researching [1–3].

However, an inherent property of these systems was quickly found i.e. hallucinations [4]. These refer to instances when LLMs generate false and misleading information that sounds plausible [5]. In order to mitigate this property, we hypothesize that one first needs to have expertise in the field they are communicating about, with the LLM. This way, the issue of hallucination can first be identified. Secondly, one needs to communicate with the AI model in an effective manner, otherwise known as prompt engineering. Moreover, prompt engineering could be used to also limit the amount of hallucinations that are happening by preventing them.

As such, before widely adopting this technology in the job market, it is important to observe whether students are equipped with the right skill set to tackle potential issues similar to the one previously described. Moreover, due to the rapid evolution of this technology, AI companies are seemingly marketing the idea that LLMs can even replace junior Software Engineers in the near future. As such, we are interested in finding out the answer to the question:

> To what extent can software engineering skills translate into effective AI usage performance compared to business students in a real life engineering scenario?

To better approach this question we can decompose it into the following sub research questions:

- Does prior SE domain expertise measurably improve iterative prompt engineering (catching hallucinations and refining prompts more effectively)?
- How do Software Engineering students compare to Business Administration students when using AI tools to solve task in a real life work setting?
- What does this imply about the need for a specialized Prompt Engineering course in SE programmes?
- What does this imply for the future job market of Software Engineering and how can universities adapt?

We believe that by answering this question, it is possible to provide some valuable insight to the Academia Environment, particularly answering the question whether Software Engineering majors need training that targets prompt engineering skills development. Moreover, by comparing the skills with Business Administration majors, we aim to also identify to some extent, how true the claims of these large AI companies are, when they promote the idea of replacing human software engineers.

Chapter 2

Background & Related Work

2.1 LLMs fundamentals

First, it is imperative to understand how LLMs work. Johnson and Hyland-Wood provide a comprehensive overview of this in their paper titled "A Primer on Large Language Models and their Limitations". They detail architectures of different AI models, training methodologies and adaption techniques.

Primarily, there are three different models: Decoder-Only, Encoder-Only or hybrid Encoder-Decoder models. Our analysis will include one of the most widely used Large Language Models on the market ChatGPT o4-mini model by OpenAI [6]. As such we will focus on the Decoder-Only type. These are optimized for text generation tasks. They achieve this by predicting the next word in a sequence. Here also lies the inherent property of LLMs to write code, as that is also a form of text. As such, they have been progressively used more in Software Engineering tasks like Code Generation, amongst other applications.

The negative property inherited by these models due to their generative nature should also be noted. Since the problem of predicting the next word in a text sequences is thought of by the neural network of the LLM as a classification problem, the answer is provided by the class probability of the word. As such, this technology is susceptible to hallucinations. These represent text or code generation that looks plausible but is factually incorrect. One potential take is that generative AI will continue to need the supervision of human engineers. However, work is done in recent years to limit the amount of hallucinations. As such, this problem can also be seen simply as an element of assumed risk when employing the model in a certain domain.

Before being released to the market, LLMs also undergo self-supervised learning on vast amounts of texts [4]. This enables them to learn language patterns, structured and semantics. Finally, LLMs are fine-tuned depending on the actions they need to perform, in the case of ChatGPT seemingly acting as an assistant and receiving instructions.

After these models are deployed they can be further refined for specific tasks by individuals through prompt engineering to produce optimal outputs [4]. Moreover, we hypothesize that engaging in iterative conversations with the model where clear feedback is provided might also reduce or prevent the appearance of hallucinations.

2.2 LLMs in SE

The software development life cycle is a well structured and documented process used by teams of developers and organizations to design, develop, test and maintain high quality software. The phases include: Requirements Engineering (RE), System Design (SD), Implementation (Coding), Testing, Deployment and finally Maintenance. Software Engineers were fully responsible of these phases. However, ever since the launch of this emerging technology, AI has already impacted some of these areas.

In RE, Marques *et al.* have conducted an literature review of how ChatGPT can be integrated into the process. Some of the most notable benefits include enhanced brainstorming and exploration, continuous learning through constant feedback and minimized human error as it automates documentation and reduces inconsistencies. However, bias in the training data of the AI model can lead to inaccurate or discriminatory requirements. Additionally, as we previously stated, hallucinations of the model can generate incorrect or misleading information that is not in agreement with stakeholder interests. Moreover, with recent models thought of being capable of in-context scheming [7] the LLM might even generate false requirements such that it achieves a certain desired outcome. Thus, the model is also able to introduce potential security vulnerabilities. In order to overcome these challenges, Marques *et al.* propose better prompt engineering to optimize prompts for more accurate and context-aware requirements generation, amongst side other improvements.

Design is a phase that requires great in-context critical thinking such that an optimal solution that achieves the desired requirements is put into practice. Generative AI however, seems to have a limited ability to innovate [1]. LLMs being probabilistic models generate solutions by sampling learned data. As such, they excel at generating software solutions that align with past human created patterns. At best the AI model is simply able to combine existing ideas to build a new software architecture. Even in that scenario human intervention is necessary to account for hallucinations, or other performance or quality aspects.

When it comes to Implementation an analysis of ChatGPT generated code suggested accuracy drop as task difficulty increases [3]. Out of 4,066 code snippets across 2,033 different programming problems, 67.8% were correct, 26.6% produced incorrect outputs and 4.4% had compilation runtime errors. The identified influencing factors are: program difficulty, programming language used, knowledge cutoff date and program size. The failures due to knowledge cutoff date seem particularly interesting as this poses the question if AI is overly dependent on the data it is trained on. In a study concerning the validity of the results provided by the SWE-Bench benchmark for various popular models, Jimenez *et al.* explain that Data Leakage amongst other issues resulted in inflated scores of the models tested when solving GitHub Issues. Data Leakage is classified as information that the LLM has seen during its training before the cutoff date. Liu *et al.* also identified in this analysis of ChatGPT significant Code Quality Issues, with a reported 47.5% of solutions having poor code style and structure, amongst other maintainability issues. The AI model also demonstrated a limited ability to correct itself. Using static analysis tools and runtime errors it managed to improve code quality by over 20% however there are still opportunities for improvements. These findings seem to suggest that human supervision is still required in the Implementation phase.

Siddiq *et al.* investigate how effective Large Language Models are in generating unit tests for Java programs. They analyze Codex, ChatGPT 3.5 turbo and StarCoder using two benchmarks (HumanEval and EvoSuite SF110). Findings suggest that the Codex model achieved 80% code coverage when it was tested on the HumanEval dataset. On the other hand, all the models struggled with the EvoSuite SF110 dataset with none achieving more than 2%. This raises two primary concerns. First, the popularity of the HumanEval dataset in the LLM research community may have influenced the training data for some of the models, thus inflating the scores significantly. Secondly, authors suggest there is also an issue with the quality of generated tests, as some of them are empty or duplicated. It should be noted that the performance reported in this study probably increased with more recent models, however it still seems human oversight is required in this process as well.

2.3 Prompt Engineering Techniques

Finally, to conduct a prompt engineering study we must first define what this concept is. Additionally, we also need to understand what classifies as a prompt in order to further define requirements for our experiment.

> Prompt engineering is the process where one guides the generative artificial intelligence (generative AI) solutions to generate desired outputs. Generative AI requires detailed instructions to create high-quality and relevant output [10].

In the AWS guide, they go on to explain that for effective prompt engineering you have to choose most appropriate formats, phrases, words and symbols. AI responses will improve through creativity of the written prompts plus trial and error. There are 8 different prompt engineering techniques described in the guide, however we will focus on the ones most appropriate for our purposes.

Chain-of-thought prompting

Chain of thought prompting requires the user to break a problem into smaller parts that resemble a thought chain. By doing so, this helps boost the models reasoning ability by having it solve the smaller intermediate problems created by the user.

The guide goes on to suggest multiple runs of the same problem, and picking the most commonly reached conclusion, only intervening when the answers are significantly different [10].

In a Software Engineering environment an example of chain-of-thought prompting could be: presenting the final goal to the model (the launch of a new software product) and then asking the model to go through the phases described in the software development life cycle (eliciting requirements, creating a design, product development, etc.).

Tree-of-thought prompting

This is a generalization of the previous method of prompting. Here the model is asked to also generate possible next steps and then prompted again to solve each of the possible next steps using a tree search method [10]. Following the same scenario as before, instead of providing the model with the development phases you would ask it to explicitly think what some next steps would be to get closer to the goal of launching the product). As stated before this more likely applies to a wider range of problems where knowledge of how things should be implemented is limited.

Maieutic prompting

Maieutic prompting focuses more on having the model answer a question with an explanation. Such that the model is then prompted again to explain parts of the explanation [10]. Applying this to Software Engineering, this style of prompting could be very useful in debugging, more specifically, it could help engineers understand exactly why an issue appeared by going on the error trace.

Least-to-most prompting

This prompting style focuses on making the model list all the subproblems of a problem and then is asked to solve them in order. This approach works well for mathematical questions however, it also has applications in the Software Engineering field [10]. By having the model solve issues step by step engineers can catch errors or hallucinations and pin point exactly where the model failed.

Self-refine prompting

Self-refine prompting refers to the instance when a model is prompted to critique its own solution. the example provided in the AWS guide presents a user prompting a model to write an essay. After the essay is created, the model is asked to critique it (i.e. lack of specific examples) and then rewrites the essay to include specific examples [10]. This approach is rather similar to how coding works. Engineers create a solution, if errors occur, they are analyzed and then a new solution is created which fixes the error. Which is why LLMs are somewhat successful at solving programming tasks, as engineers ask them to solve the problem, and when the solution proposed results in an error, the error is fed back to the LLM which then proceeds to fix it, only the error is found by a feedback tool rather then the LLM itself.

Directional-stimulus prompting

Finally, the directional-stimulus prompting technique includes hints that may be required by the model to complete the task, such that it guides it towards a desired output [10]. Focusing on Software Engineering, an example of this style would be providing the model with the tech stack required by your solution. This is very important as the model might suggest you coding solutions in a completely different programming language than the one you are required to use.

2.4 AI adoption rates

The reason the primary focus of this study is how Software Engineering skills translate into effective prompt engineering is due to the wide adoption of AI tools in the software development life cycle.

One survey conducted in May 2024 on the popular platform Stack Overflow reports 76% of respondents are using or plan to use AI code assistants. Moreover, the primary code assistant used is ChatGPT with a significant 84% usages, followed by GitHub Copilot with 49% [11], hence the use in this study of OpenAI's new o4-mini reasoning model which according to the company its targeted use is coding, science and math [6].

On another note, in a survey consisting of 1,363 participants by McKinsey & Company approximately 72% of respondents adopted AI in at least 1 business function. More specifically, 65% confirm the use of generative AI like ChatGPT. In the Healthcare industry, a comprehensive survey published in October 2024 regarding adoption of AI tools in European laboratory medicine recorded that only 25.6% of laboratories have ongoing AI projects out of 195 considered responses [13]. The primary challenges identified were significant gaps in necessary digital infrastructure and training. Finally, in the Legal industry, 51% of firms report adopting AI based-tools and solutions with 12% of them stating they are exploring this area [14].

As it can be observed, the largest adoption rates of AI tools recorded are in the Software Engineering and Business/Financial industry which is one of the primary reasons this analysis will be conducted on engineering and business graduates. Additionally, we could make the argument that these two industries are often interlaced as there are many tech centered business that aim to deliver software products or services.

2.5 Literature Review Reflection

Out of the reviewed papers, it is clear AI is slowly being integrated in the Software Engineering industry, however there are justified concerns of quality assurance, security, ethics and sometimes even performance. While work is done to improve on these aspects, human engineers experience seems to be required. Many of the studies propose prompt engineering as a way to provide quality feedback to the AI which will optimize it for the required task and tackle some of the issues, including hallucinations. Hassan *et al.* propose a goal driven approach of AI pair programmers. They are to engage in iterative conversations which would align the models solutions closer to developer and organizational goals by providing constant feedback.

Chapter 3

Study Design

This chapter will outline the steps taken to conduct an experiment of two groups of individuals (Software Engineering and Business students) in an effort to analyze their ability to solve tasks and challenges present in a real life Software Engineering environment.

3.1 Setup

The study consists of **6** participants evenly spread across the two groups. These two groups were chosen as they have the two biggest AI adoption rates across the different working industries. Participants from both groups were selected based on their willingness to participate.

The experiment was conducted twice, on June 22nd and June 28th for the Software Engineering and the Business Administration group respectively. Before starting the experiment, a privacy notice was provided to the participants informing them of the data being collected and their rights to object to data processing and data storage. After reading the notice, participants were instructed to study the experiment document in detail which contains information regarding the technologies they are allowed to use, setting up the scenario, and explaining the tasks they have to complete. This document was provided in both physical and digital form to the participants (see **Appendix A**). After reading the document, participants began working on their tasks for a maximum duration of 3 hours.

3.2 Experiment

The experiment requires students to solve 3 tasks that are part of a simulated Software environment with the help of the ChatGPT o4-mini reasoning model [6]. The environment consist of a mock-up E-Ticket web application with a Graphical User Interface (GUI), a server and a database. The 3 tasks were designed to target one or both groups. More specifically, one task is neutral (minimal coding knowledge needed), one targets the Software Engineering group an one focuses on Business concepts.

The neutral task consists of creating an HTML form with the intent of adding a new ticket. All the dynamic JavaScript functions and server calls were created beforehand, thus students were simply required to create the form and use the appropriate input and button id fields. This task was designed with minimal coding knowledge requirements in mind as business students had the opportunity to leverage the AI model as a coding assistant. This task should not pose a significant challenge to the model as previously reviewed predecessors models to the o4-mini demonstrated promising performance on simple coding challenges (see **Appendix A**).

The second task, targeting Software Engineering students, consists of implementing the *delete* ticket operation. The students were required to create the desired functionality for the button available on the GUI, the JavaScript server request, and finally the server logic handling the request, whilst respecting the server's layered architecture. This task was designed to examine the ability of participants to efficiently navigate a development environment and link multiple components with one another. Due to the increased complexity, it is also likely that the AI model will not simply provide the correct answer as its performance is negatively correlated with problem complexity (see **Appendix A**).

Finally, the third task targets Business students. One concept Master of Business Administration students are expected to know by the end of their studies is dynamic pricing. As defined by the Harvard Business School, "dynamic pricing is a strategy that bases products or services on evolving markets trends, such as: supply and demand, competitor pricing and inventory levels" [16]. Thus, participants were given the following formulas and other necessary data, to compute the new price of 3 different tickets depending on the number of days left till the event and ticket availability:

1. Dynamic Price:
$$P_{\text{new}} = P_0 \times \left(1 + \text{markup}_{\text{dyn}}\right) \times \text{seat type}$$
2. Dynamic Markup:
$$\text{markup}_{\text{factor}} = c \times \left(1 - \frac{V_t}{V_0}\right)$$

where:

- P_0 - Base price
- $\text{markup}_{\text{dyn}}$ - Total markup
- seat type - VIP/Standard
- c - Max markup constant(%)
- V_t - Current variable
- V_0 - Initial variable

To link this task back to the Software Engineering environment, the students were asked to update the prices by creating an SQL query in the Database, something that again should not pose a significant challenge for the AI model as it is a simple coding task (see **Appendix A**).

During the experiment the participants had access to a computer using VSCode as a coding editor, a ChatGPT conversation window, DB Browser for Sqlite for database browsing and interaction and finally Postman Desktop app to test the API endpoints if needed.

It should be noted that participants were allowed to edit the code themselves. While this introduces a bias towards Software Engineering students due to the nature of the environment and tasks, it will help answer the questions whether domain knowledge impacts task success rate. Moreover, it will be evident from the recorded conversation with the model to what extent the participant engaged in conversations with the AI. Thus, the question focusing on prompt engineering skill will also be answered.

After students finished each task, the total time elapsed since the beginning of the experiment until that moment was recorded for further quantitative analysis. After finishing the experiment, their platform implementation and conversation with ChatGPT were also documented for quantitative and qualitative analysis respectively. Below is a figure highlighting this experiments outline (see **Figure 3.1**).

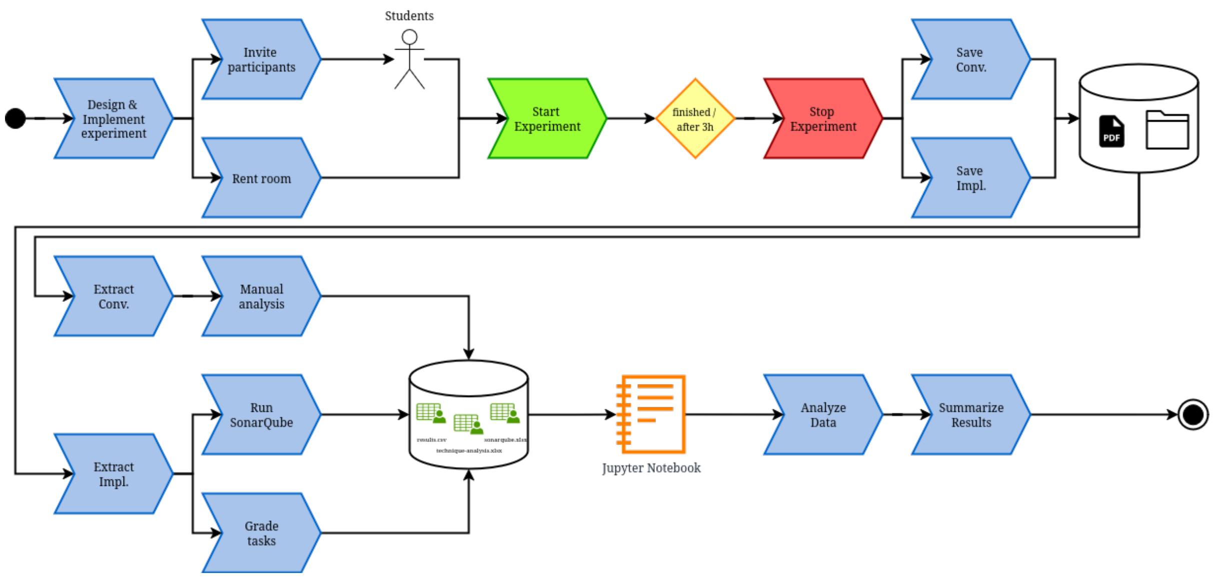

Figure 3.1: Study Design diagram

3.3 Controlled variables

To ensure solutions from one student would not "bleed" into other chats, persistent memory between chats has been turned off. Additionally, to restrict students from accessing the internet or having conversation outside the scope of the experiment, internet access has been turned off from the account settings and the models has been given a system prompt to ensure it only answers to questions belonging to the Software Engineering and Business Administration topics. The prompt given was designed in accordance to the guide available on **mustvlad** GitHub repository [17]. Below is the exact prompt used during the experiment:

> [SYSTEM] You are an AI model that specializes in the Software Engineering/Development and Business sectors. Only respond to questions within these two domains. If a user asks about unrelated topics politely decline and remind them that you are only able to help with issues related to these domains. You are part of a controlled experiment where users are asked to solve 3 tasks part of a real life work Software Engineering environment. Users will interact with you in order to solve these tasks. Your job is to aid them in this process, whether that is through explaining concepts, answer questions or help them solve problems.

3.4 E-Ticket web app

We chose to develop our own platform as it is unlikely for the AI model to have seen a similar application before. Thus, we reduce the possibility of model knowing the answer to user questions from the training data, which would have been likely if coding challenges from a popular platform like LeetCode would have been chosen.

As previously described, the platform consists of 3 major sections (client or GUI, server and data). The server was developed in Python using the Flask for a developmental server. It followed a basic layered architecture consisting of the API layer, to implement the endpoints the Client would call upon, the Business layer, to implement the logic behind the server request, the Data layer, which would enable the server to communicate to the database and finally the Model layer which contained the models required by the implementation. While solving their tasks, students were expected to follow this layered architecture and were graded accordingly.

3.5 Analysis & Grading

Students will be analyzed according to their implementation for quantitative analysis and according to their chat with the LLM for qualitative analysis.

To asses the former, the initial implementation of the platform was analyzed on the SonarQube tool. SonarQube is a popular open-source code analysis tool that provides insights into a project security, reliability and maintainability, amongst other metrics such as code duplication and cyclomatic complexity [18]. We recorded the original issues found by the tool which stem solely from the methods that students are required to implement both on the Client and Server side. Thus, if students followed the correct platform architecture and followed code security, reliability and maintainability guidelines, all these issues should disappear leading to no issues being detected.

From the documented implementations, the projects will be passed through SonarQube to detect changes in the issues list that may have been introduced by the AI model or by the user. In addition, each students implementation will be graded based on defined rubrics. To see these rubrics please refer to Appendix B. These will be documented and together with the recorded times for each task will make up for the quantitative analysis of this thesis. Additionally, the recorded conversations will be examined manually to observe whether prompt engineering techniques previously listed in the literature review which were potentially used by the participants.

Chapter 4

Results

This chapter will highlight the results of our experiment. There will be three primary sections showcasing students task performance, their chosen implementation and finally their conversation with the o4-mini model.

4.1 Student task performance

Four scatter plots were utilized to showcase students performance for each of the 3 tasks of the experiment and for the overall experiment in the time and grade space. On the x-axis we have the time it took a student to solve a task in H:MM:SS format and on the y-axis we have the grade the student received according to the rubric in the [0, 2] range with 0 representing a fail, 1 representing a pass and 2 representing a perfect score (see **Appendix B**).

Figure 4.1 showcases most of the students from both groups finishing task 1 in approx. 15 minutes with a Software Engineering outlier shortly after 30 minutes. When it comes to their task grade, students part of the Software Engineering group scored 1.75 or above, whereas students in the Business Administration group scored between 1 and 1.5. In **Figure 4.2** it can be observed that Software Engineering students finished task 2 under 30 minutes with one student finishing it under 15 all with scores of 2. On the other hand, Business Administration students finished task 2 in between 1 hour and 15 minutes and 1 hour and 30 minutes with one of them finishing it slightly over 1 hour. Their task grade as before ranged between 1 and 1.5. Task 3 performance was plotted in **Figure 4.3**. Business Administration students all finished their tasks below 15 minutes, whereas Software Engineering students completed it between 15 and 30 minutes. With the exception of one Software Engineering student who scored 1.5, the two groups obtained a task grade of 2. Finally, the overall performance of each individual student for the entire experiment has been plotted in **Figure 4.4**. Two Software Engineering students finished the experiment in a total time slightly under 1 hour with an average grade of 2 for all the experiments. One other student part of the same group finished slightly under 1 hour and 30 minutes with an average grade of 1.75. When it comes to Business administration students, two finished their tasks around the 1 hour and 45 minutes mark with another student finishing slightly above 1 hour and 30 minutes. Their average experiment grade ranges from 1.5 to 1.66.

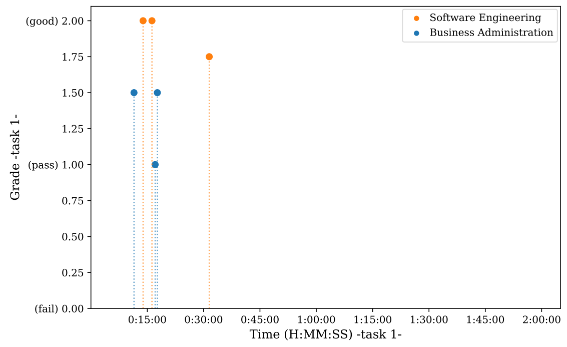

Figure 4.1: Completion time and grade distribution by study program for Task 1

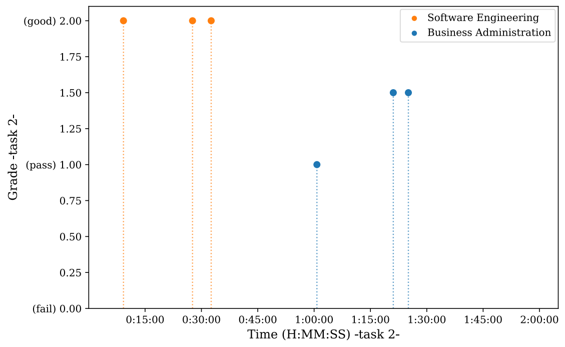

Figure 4.2: Completion time and grade distribution by study program for Task 2

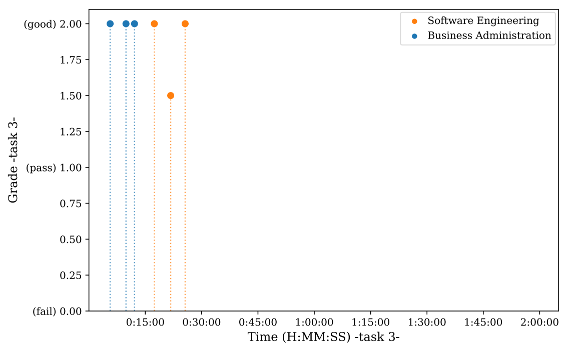

Figure 4.3: Completion time and grade distribution by study program for Task 3

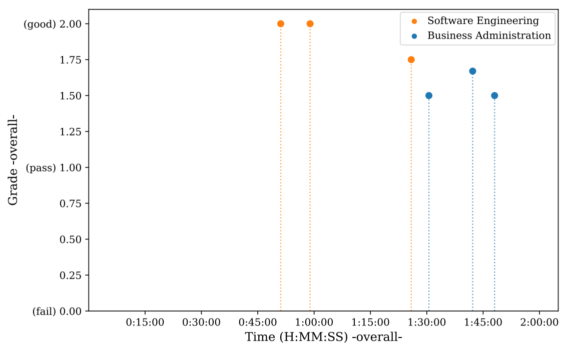

Figure 4.4: Overall completion time and grade distribution by study program

Two bar charts were plotted to showcase differences between the two groups more clearly. The values represented on the y-axis in each graph are metric averages belonging to the students within the group. In **Figure 4.5** we can observe the average completion time for task 1, 2, 3 and overall of the two groups. The Software Engineering group finished all tasks with and average between 15 and 30 minutes. The Business Administration group on average solved the first task slightly above 15 minutes, the second task just above 1 hour and 15 minutes and the third task well under 15 minutes. For the entire experiment. Software Engineers managed to finish it on average between 1 hour and 1 hour and 15 minutes, whereas the Business Administration students completed it on average between 1 hour and 30 minutes and 1 hour and 45 minutes. **Figure 4.6** depicts the average grade for task 1, 2, 3 and the average experiment grade for each group. Across all tasks Software Engineering students scored between 1.75 and 2 with an overall group average within the same range. On the other hand, Business Administration students scored on average the same grade for task 1 and 2, just above 1.25 with a score of 2 for task 3, resulting in an overall group average just above 1.5.

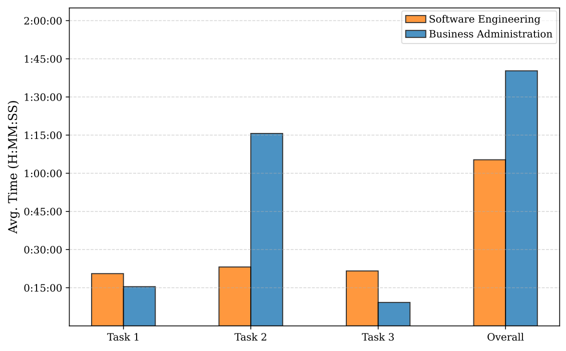

Figure 4.5: Average completion time by task and overall, grouped by study program

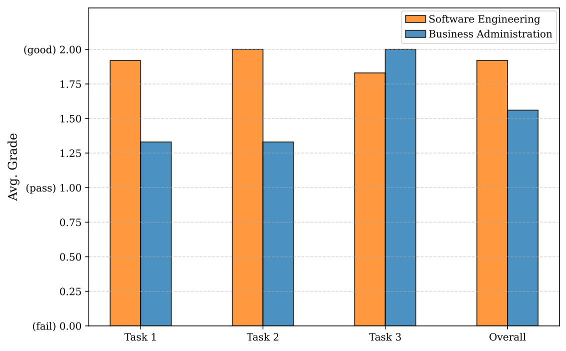

Figure 4.6: Average grades by task and overall, grouped by study program

Finally, to clearly view each of the groups strengths in this experiment another bar plot was created to showcase relative performance of each group for each task and for the entire experiment. To achieve this a performance metric for each student was calculated based on the average between the normalized grade and time values recorded from which a performance group average was computed later. It should be noted the *MinMaxScaler* scaler was used from the scikit-learn python library. This normalization method utilizes the minimum and maximum of the dataset to scale the results between the 0 to 1 range as opposed to predefined ranges. As such, the plotted performance for each group is relative to each other. This was done to clearly highlight the strengths of each group based on their implementation. For task 1 and 2 the average performance of the Software Engineering group is higher then that of the Business Administration group, however for task 3 this is reversed (see **Figure 4.7**).

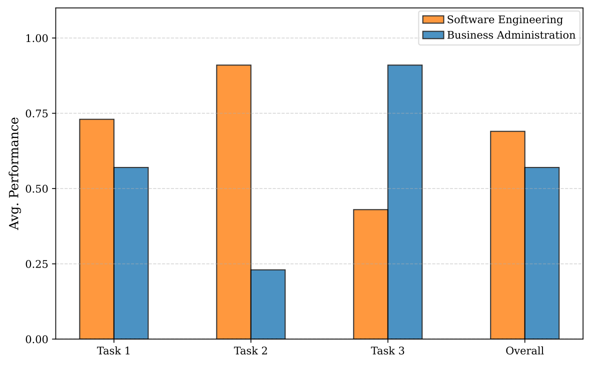

Figure 4.7: Normalized performance, grouped by study program

4.2 SonarQube implementation analysis

A parallel coordinates plot was used to map out each of the projects according to SonarQube analysis. It should be noted for Security, Reliability and Maintainability, scores were computing by summing all the severities of the issues present in the project for each respective category. Each of the projects analyzed have been color coded based on the group they belong to. As seen in **Figure 4.8**, there are no recorded security and reliability issues for all projects analyzed. When it comes to maintainability, all students improved on the initial platform implementation with 2 of them (one from Software Engineering one from Business Administration) reducing the issues all the way down to 0. For all projects there was a total of 4 security hotspots (including the base project), and all managed to keep the code duplication percentage at 0. For cyclomatic and cognitive complexity all projects submitted by the students scored lower than the initial implementation.

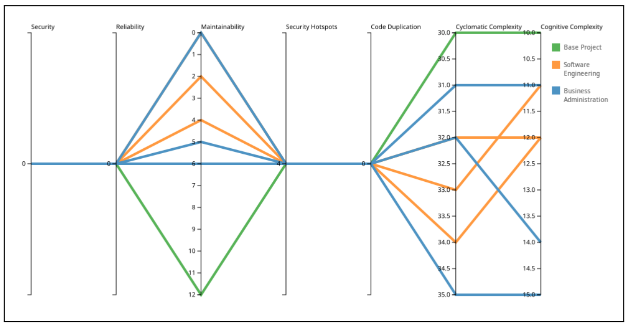

Figure 4.8: Comparison of software quality metrics between projects, grouped by study program

4.3 Student conversation analysis

Student conversations were manually analyzed. First we looked for user quotes inside their prompts that would constitute any of the prompting techniques described in 2.3. As we are interested solely in the student's prompting capabilities, any copy-pasted task descriptions were ignored. As some of the techniques were employed with varying quality by the study participants, a degree was also assigned to the quotes identified. These quotes can be observed in **Appendix C**. Then, an overview of the conversation style between the students and the LLM was created, and coupled with the previous analysis, resulted in the summary of findings in **Table 4.1**.

Student	Summary
1-se	Student prompts were direct and focused solely on obtaining code snippets. No context was provided to the LLM regarding the experiment, the environment or the tasks. Through its targeted requests the student demonstrated "Directional-Stimulus Prompting", however showed no evidence of any other techniques. The conversation style suggests confidence in solving the tasks but a limited effort to leverage the model's capabilities.
2-se	The student provided extensive details about the environment and the tasks at hand to the LLM, treating it as a collaborative partner. Segmented tasks for the AI model, thus demonstrating "Chain-of-thought Prompting". Additionally, when unsure, the student prompted the model to identify missing code components and refine its solutions demonstrating "Least-to-Most" and "Self-refine" prompting techniques in addition to a strong desire for accuracy and deeper understanding. Finally, when requesting solutions, they provided accurate details regarding the output format, such as programming language used or application architecture style demonstrating "Directional-Stimulus Prompting".
3-se	Participant provided occasional project context demonstrating some degree of "Directional-Stimulus Prompting". When issues appeared they relied on the model to troubleshoot and potentially solve the problem on its own. This hints to some degree of "Maieutic Prompting". The student often provided the entire code file to the LLM which led to the model reviewing its own code based on provided suggestions. This could be interpreted as some degree of "Self-refine Prompting". Overall they displayed enough technical knowledge of the field to ask the correct questions or point the LLM to the right directions to solve the tasks, however they over-relied on the ability of the LLM to solve the tasks on its own.
4-ba	The student actively engaged with the LLM throughout the entire experiment by often providing an exact copy of the task or pasting an entire code file. They frequently requested clarifications on how and where to integrate code, showing a lack of knowledge concerning Software Engineering environments. However, by being unsure, the student did engage in debugging sessions with the model (in the lead) thus, demonstrating a small degree of "Maieutic Prompting". Additionally, because of the same reason, during its debugging sessions, the student also prompted the AI to review some of its solutions, demonstrating a small degree of "Self-refine Prompting". Overall, the conversation flow suggests a lack of field knowledge and a dependency on the LLM both for coding support and foundational understanding of topic fundamental to web development.
5-ba	Actively engaged with the model by providing code files and copy-pasted task instructions. However, they did prepare the model to receive the instructions and asked for guidance to solve it, demonstrating some degree of "Least-To-Most Prompting". When issues appeared, the student relied on the LLM to troubleshoot, however they provided a detailed description of the events that led up to the issue, thus aiding the process. This is consistent with and intermediate degree of "Maieutic Prompting". Finally, they also demonstrated some degree of "Chain-of-thought Prompting" by providing a large overview on how task 3 should be solved. Overall, this shows a reliance on the LLM for coding and working with an unknown environment. Frequently sought validation from the LLM indicates low confidence and uncertainty which points to low expertise in Software Engineering.
6-ba	Structured the integration with the LLM in a staged, step-by-step manner, by providing all the instructions and context before asking for help. This is consistent with some degree of "Chain-of-thought Prompting" and a basic level of "Least-to-Most Prompting" due to the lack of field knowledge. This is further confirmed as the student frequently asked for the tasks to be broken down into smaller and smaller parts and required repeated explanations and validations from the model. Moreover, multiple times due to this confusion they asked the LLM to start over, making the process of finishing the experiment harder. Overall, the student relied on the LLM extensively both as a tutor and as a technical guide.

Table 4.1: Student conversation summaries

Chapter 5

Discussion

In this chapter, we discuss the results of our experiment starting with the student task analysis, then moving onto the analysis of their implementations and finishing with the insights gained from the conversations with ChatGPT o4-mini model.

For task 1, students had to create an HTML form using the correct components IDs such that the insert ticket functionality is completed. As it was revealed by our analysis, both groups solved the task in roughly the same amount of time $\approx$ 15 minutes with one Software Engineering student finishing around the 30 minutes mark. However, when it comes to grades, the Business Administration students obtained lower grades due to their implementation of the form (see **Figure 4.1**). As described in **Appendix B** while they did provide a working solution which successfully submitted a ticket on the platform, styling was inconsistent with the platform. In one case, the form was displayed as a completely separate component which looked imported from another website. Similarly, one Software Engineering student also had their grade penalized as a left-over HTML header from ChatGPT generated code was found during implementation analysis. However this was classified as a small inconsistency as it did not break the website flow.

Based on the reviewed conversations with the model, students from both groups made a request to the LLM to generate an HTML form. However, Software Engineering students either applied their domain knowledge or worked more extensively with the model to provide a more appropriate solution to task 1. On the other hand, Business Administration students obtained the initial solution and implemented it, with no observed desire to validate or perfect it. Moreover, they asked clarifying questions to identify where they are supposed to write the generated code or had the AI provide a complete implementation of the *"index.html"* file.

For task 2, students had to complete the DELETE ticket operation starting with the server request, and finishing with the request logic implementation on the server. As it was revealed in **Figure 4.2** Software Engineering students finished this task in approximately 30 minutes, with one student completing it under 15, whereas the fastest Business Administration student took $\approx$ 1 hour to finish the task. The other two students took between 1:15:00 and 1:30:00 to complete it. SE students all received a perfect grade as their implementation followed the platforms architecture and made use of the helper functions and modules left behind from the other ticket operation. On the other hand, while BA students 4 and 5 roughly respected the platform's initial architecture, they failed to use the *"execute_query()"* function already provided in the *"SqlDataService"* implementation. Moreover, student 6 did successfully implement the functionality required for a pass, however completely ignored the platforms architecture and created an SQLite connection in the *"api.py"* module, which should be responsible for defining endpoints and request validation only.

The AI conversations again provided significant insight into the workflows employed by the students to solve this task. Again, the Software Engineering group perfected the AI generated code using their extensive domain knowledge, whereas Business Administration students asked for detailed steps to help clarify what code to write and where to write it. All of them started by copy-pasting the task description into ChatGPT and asking it for help. When the model generated some code to be added into the platform, the students were confused as to where they should add this code. BA participants 4 and 5 started asking questions to clarify the platform architecture, and as soon as they validated with the model they are in the correct place, they provided it with the code with the intention to modify it. As such, the model gained more context regarding the platform and was able to provide mostly correct suggestions and answer the clarifying questions posed by the students. While the model occasionally assumed the implementation was written in a different programming language (i.e. server written in Express.js), it was able to correct itself once the context was clear enough. On the other hand, the BA student 6 had a different approach. They provided all the necessary context at the start of the conversation. While this is not a bad practice, the model provided an in-depth guide to validate that the platform's implementation is as described in the experiment instructions. As such, the student got stuck on trying to identify whether the ticket ids were correctly configured in the database schema. Eventually, after starting over multiple times with task 2 the student started completing it from the server side first as opposed to the other two. With little to no context about the platform's architecture the LLM provided generated code to be placed in non-existent files which confused the situation even further. Eventually, through trial and error, the student managed to complete the task, however they were penalized for their implementation. It should be noted, that BA students also had difficulty in debugging issues due to their lacking Software Engineering domain knowledge. More specifically, they failed to notice the server shut down after code syntax issues from faulty copy-pasting, which resulted in significant time lost to first understand the server stopped and then figuring out how to restart it.

For task 3, students had to update three ticket prices based on a mathematical formula meant to simulate dynamic pricing. As it can be observed in **Figure 4.3**, Business Administration students solved it with perfect scores and under 15 minutes, whereas Software Engineering students took between 15 and 30 minutes to finish the task and achieved a perfect score with the exception of one student that had a small calculation error. More particularly, their implementation failed to consider the VIP status of ticket A. As opposed to task 2, we observed a somewhat insignificant difference between the two groups. This is further confirmed from the conversation analysis.

With the exception of one student from the SE group that manually calculated the prices, it appears that all used the AI model to solve this task. With no further validation, the BA students applied the AI generated solutions after a few clarifications regarding the Database Viewer platform usage. Similarly, SE student 3 applied the same solving method and thus failed to validate the price of ticket A resulting in the lower grade. Software Engineering student 2 had more accuracy oriented approach where it reasoned together with the LLM on the correct interpretation of the task description. Thus, they were able to calculate and update the price correctly. Based on the workflow used by most students it seems the AI model was able to solve the task quite easily resulting in the short solving times and high grades. In the case of student 3 from the SE group, the conversation was quite long (119 pdf pages compared to the average 40-50 pages of the other students). One potential explanation for the incorrect calculation of ticket A could be the diminishing context length, however since the context for task 3 was recently provided and the student did not quote something from the beginning of the conversation this is somewhat unlikely. Given this analysis of the students workflow, the observed increased performance of the Business Administration group is likely due to outsourcing the whole task to the AI model with no validation on their part, whereas SE students seemed to place more value on the accuracy of the output. This led to faster task completions and thus increased overall performance, however questions should be raised regarding the degree of trust placed in the model to solve the task correctly. And, as this experiment has shown with student 3 from the Software Engineering group, these questions are justified.

As it was observed in **Figure 4.5** Software Engineering students had a more consistent solving time then Business Administration students. This likely happened due to the increased domain knowledge and familiarity when working with the given tools, thus reducing their overall average time in solving all three tasks by approximately 30 min. While all students managed to solve the tasks, **Figure 4.6** shows higher average grades of Software Engineering students for task 1 and 2, with Business Administration students scoring slightly higher on task 3. As **Figure 4.7** plots the average relative performance (time and grade) of the two groups for each task and the entire experiment, we observe SE students performed slightly better on task 1, significantly better on task 2 and worse then BA students on task 3. Overall the task analysis suggests:

Finding 1: SE students outperformed BA students in this experiment of a simulated Software Engineering environment. BA students had difficulty in navigating the platforms architecture and working with the provided tools. As such they utilized ChatGPT as a guiding tutor. On the other hand, SE students had a more accuracy oriented approach (seemingly aware of the model's potential limitations) and thus worked together to solve tasks, with them often taking the lead.

When it comes to the SonarQube implementation analysis, the results recorded could not be easily differentiated in the platform, due to their implementation of calculating metric scores [18]. As such, the metrics had to be extracted from the platform in raw form and slightly modified for our analysis.

More precisely, the security, reliability and maintainability scores were calculated by summing the impact levels of all issues detected by SonarQube in that category. The rest of the metrics recorded were not modified.

Figure 4.8 reveals no security or reliability issues found in the students' implementations, regardless of using ChatGPT generated code or not. When it comes to maintainability, all participants improved on the initial implementation with 2 students one from each group reducing the total score down to zero. For the rest of the students, the two Software Engineering participants slightly outperformed the remaining Business Administration ones. An in-depth analysis of the detected issues revealed SE students had lower number of issues and less critical (i.e. *"Rename this variable; it shadows a builtin."*). On the other hand, BA student 6 contained two of the four initial issues the Base Project had due to their implementation of task 2 where they established a Database connection in the *"api.py"* file, thus having 2 left-over not-implemented functions.

All projects, including the base one contained 4 security hotspots. As it was previously mentioned this platform was a mock-up meant solely for this experiment, thus it was not built to be production ready.

The code duplication recorded for all projects was also 0.0%. This can be explained given the small code base the students had to modify < 1000 LOC (lines of code). For cyclomatic and cognitive complexity, it should be noted the recorded values represent the total sum of all complexities in the project. Further analysis in the SonarQube portal revealed a max cyclomatic complexity per unit ≤ 10 for all students except student 4 from the BA group with a recorded complexity of 11. According to Heitlager *et al.* the recorded measures of code duplication and cyclomatic complexity indicate a low level of risk and good project maintainability [19]. Finally, cognitive complexity slightly varies between the different students' implementations. Software Engineering students managed to produce slightly more consistent results with overall lower cognitive complexity then Business Administration students. Overall, the SonarQube analysis suggests:

Finding 2: On average SE students slightly outperformed BA students and were more consistent with respecting the platform architecture, leading to the decreased maintainability scores. Additionally, as students utilized ChatGPT generated code in their final implementation, it seems it did not introduced any new security or reliability issues. We believe this and the comparable performance between the two groups can also be justified due to the size of the mock-up platform.

Finally, the conversation analysis revealed interesting insights into how students tried to structure their workflow, including possible prompt engineering techniques used. As described in **Table 4.1** Software Engineering students demonstrated a good use of "Directional-Stimulus Prompting" as they often clearly asked the AI model to generate code according to the technologies and programming languages used in the E-Ticket platform. On the other hand, Business Administration students managed to provide some context through the copy-pasted task descriptions, however the majority of it was gained as they uploaded more platform code files to ChatGPT. This also poses a confidentiality risk.

Two out of three SE students sought to leverage the model to solve the tasks more efficiently. As such, throughout the process, we observed some prompts where they either guided the model through solving the issue or they asked the model for some guidance when unsure. However they applied their extensive domain knowledge to validate what the model was outputting, thus demonstrating "Chain-of-thought", "Least-to-Most" and sometimes "Self-refine" prompting styles. Some "Maieutic Prompting" was also observed as the students engaged in debugging, however, they more often relied on their own knowledge. On the other hand, BA students looked to the model for guidance rather then solving tasks more efficiently. As such, they asked the LLM for detailed steps, thus demonstrating "Least-to-Most Prompting". "Maieutic Prompting" was also observed as they were looking to debug, however multiple outputs were required for them to understand what they had to do to solve the problem. There are some small hints of structuring the LLM output with their own set of instructions, resembling "Chain-of-thought Prompting", however the provided steps were not significantly detailed. Overall the conversation analysis suggests:

Finding 3: Both SE and BA students demonstrated some levels of prompt engineering. SE students are seemingly more aware of the potential issues with the model when it comes to the output's accuracy demonstrated by their increased interest in validating the LLM. However, the prompting techniques used by both groups were successful more due to the ability of the model to interpret and understand the users' intentions rather then their capability to convey their ideas. Moreover in the case of BA students, questions can be raised regarding their capacity and efficiency in solving the tasks if a clear set of instructions was not provided, as they would not be able to utilize the copy-paste strategy effectively.

5.1 Threats to validity

Throughout the execution of this experiment we identified a small number of improvements we could have made. The issues primarily stem from the study design and if solved they could improve on the quality of the findings.

Firstly, participants were given an electronic copy of the instructions that they had available throughout the experiment. One behaviour we expected from some students was that of copy-pasting the direct task instructions into the AI conversation, a behaviour we did end up observing. Moreover, the tasks contained extensive technical details on how to solve it correctly and what they would be graded on. Such level of detail is not often seen in real life working environments where individuals would have to infer them on their own. If students would not have been allowed a digital copy we hypothesize they would either lose time writing the entire task into the ChatGPT prompt word by word, or we would be able to gain additional insight into their ability to prompt engineer a correct output from the model. Additionally, questions can be raised regarding the ability of said students to solve the tasks.

A different issue focuses on the third task of this experiment which was targeting Business Administration students. As we observed in the results most of the students were able to solve this task rather quickly, with little variation between groups. As we discussed we accredited this to the workflow employed by the majority of these students, more specifically outsourcing the task to the LLM. While we did observe some mistakes being made by the AI model, we hypothesize the task was somewhat simplistic to fully explore the capacity of the students to apply prompting techniques to guide the model to the correct response.

On another note, we also identified some issues during project analysis. When assessing whether students solved task 1 correctly, in one instance the form was not submitting the ticket, which was conflicting with the data collected during the experiment. After some debugging we concluded the issues was caused by the button type used in the form (submit) which when pressed was interfering with the asynchronous function written in the base project. This appeared when we were analyzing task success in a Firefox based browser. After switching to Chromium, the form submitted successfully, thus the student was awarded full marks. Alongside providing us with interesting insights into how the two different browsers executed the written code, given our study involved web development, a standard browser should have been chosen to ensure consistency.

Another issue stems from the size of the mock-up application. As it was discussed in the SonarQube analysis, student project iterations varied only slightly, as such we had difficulty analyzing project metrics as SonarQube marked all metrics with a score of A. Moreover, the model analysis would have benefited from increased project size as we would have been able to asses how the model performs with increasing complexity. As we reviewed in this paper that model performance is in negative correlation with project complexity, we would have been able to gain more insights into how students would navigate this environment and validate the LLM's output.

Finally, this experiment could have benefited from a larger number of participants. While we did gain insights into how students are able to leverage AI models to solve tasks, with a larger sample size a more accurate generalization could have been made.

Chapter 6

Conclusion

This paper aimed to analyze the ability of Software Engineering and Business Administration students to leverage ChatGPT o4-mini model to solve tasks in a simulated real life Software Engineering environment. More particularly, the focus was on assessing comparing the two groups based on task success rate, analyzing their implementations and observing how well they leveraged the model through prompt engineering techniques.

To achieve this, we ran an experiment comprised of six participants, three for each group. The students had to solve three tasks, one targeting both groups, one targeting the Software Engineering group and one targeting the Business Administration group. During this experiment they had access to the ChatGPT o4-mini model. Task success rate was graded according to defined rubrics, their implementation was analyzed using SonarQube and the conversations between the students and the model were examined manually to search for prompt engineering techniques used.

Our results suggest that on average, Software Engineering students were able to outperform Business Administration students in this experiment. Both groups sought to leverage the model to solve the tasks, however the SE group seemed more aware of the model's limitation due to their increased domain knowledge and thus had a more accuracy oriented approach. Domain knowledge also had a measurable impact onto their ability to efficiently navigate the environment. When it comes to prompts used, both groups demonstrated some levels of prompt engineering, however the success of the prompts is more likely attributed to the ability of the model to understand the students' requests. Software Engineering students did however, pose more accurate and targeted requests to the LLM, whereas Business Administration students sought guidance.

Overall, these results imply the need of Software Engineering expertise in the industry to leverage the model effectively. Otherwise, companies could risk creating code base issues with significant downtime to solve them. However, AI models do seem useful to solve tasks more efficiently under the guidance on an experienced engineer. On another note, given our model conversations analysis it seems students could benefit from a prompt engineering course to learn how to communicate their requests to the LLM more accurately.

6.1 Future work

It would prove interesting to conduct a further study analyzing how well a team of senior and junior software engineers fairs against a team of senior engineers using an AI model to develop a project. Such a study could offer more insight into how Large Language Models could be leveraged efficiently into real life working environments and what are the implications of the increasing performance of this technology on the job prospects of non-experienced engineers. Such a study could also benefit from the adaption of the solutions proposed for the issues encountered in this experiment.

Bibliography

[1] M. Rinard, "Software engineering research in a world with generative artificial intelligence," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3649399. [Online]. Available: https://doi.org/10.1145/3597503.3649399.

[2] N. Marques, R. R. Silva, and J. Bernardino, "Using chatgpt in software requirements engineering: A comprehensive review," *Future Internet*, vol. 16, no. 6, 2024, ISSN: 1999-5903. DOI: 10.3390/fi16060180. [Online]. Available: https://www.mdpi.com/1999-5903/16/6/180.

[3] Y. Liu *et al.*, *Refining chatgpt-generated code: Characterizing and mitigating code quality issues*, 2023. arXiv: 2307.12596 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2307.12596.

[4] S. Johnson and D. Hyland-Wood, *A primer on large language models and their limitations*, 2024. arXiv: 2412.04503 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2412.04503.

[5] Google Cloud, *What are ai hallucinations?* 2024. [Online]. Available: https://cloud.google.com/discover/what-are-ai-hallucinations.

[6] OpenAI, *Introducing openai o3 and o4-mini*, https://openai.com/index/introducing-o3-and-o4-mini/, Apr. 2025.

[7] A. Meinke, B. Schoen, J. Scheurer, M. Balesni, R. Shah, and M. Hobbhahn, *Frontier models are capable of in-context scheming*, 2025. arXiv: 2412.04984 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2412.04984.

[8] C. E. Jimenez *et al.*, *Swe-bench: Can language models resolve real-world github issues?* 2024. arXiv: 2310.06770 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2310.06770.

[9] M. L. Siddiq, J. C. Da Silva Santos, R. H. Tanvir, N. Ulfat, F. Al Rifat, and V. Carvalho Lopes, "Using large language models to generate junit tests: An empirical study," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE 2024, ACM, Jun. 2024, pp. 313–322. DOI: 10.1145/3661167.3661216. [Online]. Available: http://dx.doi.org/10.1145/3661167.3661216.

[10] Amazon Web Services, *What is prompt engineering?* 2024. [Online]. Available: https://aws.amazon.com/what-is/prompt-engineering/.

[11] Stack Overflow. "Developers get by with a little help from ai: Stack overflow knows code - assistant pulse survey results." (May 2024), [Online]. Available: https://stackoverflow.blog/2024/05/29/developers-get-by-with-a-little-help-from-ai-stack-overflow-knows-code-assistant-pulse-survey-results/.

[12] McKinsey & Company, "The state of ai," McKinsey & Company, Oct. 2023. [Online]. Available: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai.

[13] J. Cadamuro *et al.*, *Clinical Chemistry and Laboratory Medicine (CCLM)*, vol. 63, no. 4, pp. 692–703, 2025. DOI: doi:10.1515/cclm-2024-1016. [Online]. Available: https://doi.org/10.1515/cclm-2024-1016.

[14] Liquid Legal Institute, "First global report on ai in legal practice," Liquid Legal Institute, Mar. 2024. [Online]. Available: https://liquid-legal-institute.com/wp-content/uploads/2024/03/E-Book-First-Global-Report-on-AI-in-Legal-Practice.pdf.

[15] A. E. Hassan, G. A. Oliva, D. Lin, B. Chen, Z. Ming, and Jiang, *Rethinking software engineering in the foundation model era: From task-driven ai copilots to goal-driven ai pair programmers*, 2024. arXiv: 2404.10225 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2404.10225.

[16] H. B. S. Online, *Dynamic pricing: What it is & why it's important*, Harvard Business School, 2023. [Online]. Available: `https://online.hbs.edu/blog/post/what-is-dynamic-pricing`.

[17] mustvlad, *ChatGPT-System-Prompts*, 2025. [Online]. Available: `https://github.com/mustvlad/ChatGPT-System-Prompts`.

[18] *Sonarqube 10.5 documentation*, `https://docs.sonarsource.com/sonarqube-server/10.5/`, Accessed on August 10, 2025.

[19] I. Heitlager, T. Kuipers, and J. Visser, "A practical model for measuring maintainability," in *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*, 2007, pp. 30–39. DOI: `10.1109/QUATIC.2007.8`.

Appendix A

Experiment Instructions

Welcome!

Thank you very much for deciding to participate in this experiment!

Privacy Notice

1. Who We Are

This study is conducted by **Matei Avram**, Master's student in Software Engineering at **Universiteit van Amsterdam**. All experiment data is stored securely and accessible only to the principal researcher and study supervisors.

2. Purpose of Data Collection

We're examining how Software Engineering and Business Administration students interact with an integrated AI assistant while completing three hands-on tasks (adding a ticket form, deleting tickets, and updating ticket prices) in a simulated real life work Software Engineering environment.

3. Data We Collect

- **Program of Study:** Your academic track (Software Engineering or Business Administration)
- **AI Chat Logs:** All messages exchanged with the embedded AI assistant
- **Task Artifacts:** Final modified project, SQL statements, and related files you submit

4. How We Use Your Data

- **Analysis:** We evaluate solution correctness, prompt-engineering approaches, and SonarQube metrics.
- **Publication:** Aggregated, anonymized results may appear in the master's thesis and related academic publications.

5. No Personally Identifiable Information

We do **not** collect names, email addresses, IPs, or any direct identifiers. If you accidentally include personal details in chat, those entries will be redacted before analysis.

6. Data Retention & Deletion

- **Retention:** Raw data (chat logs & artifacts) will be kept until the completion and public defense of the master's thesis.
- **Deletion:** All raw data will be permanently deleted within 30 days following thesis defense.
- **Aggregated Results:** Fully anonymized results may be retained indefinitely for academic dissemination.

7. Your Rights

- **Consent:** By participating, you agree to this notice.
- **Withdraw:** You may withdraw at any time without penalty. To do so, notify the experiment overseer duing its duration or email `matei.avram2@student.uva.nl`. Upon withdrawal, all your raw data will be deleted within 7 days.

8. Contact

For questions, data-access requests, or withdrawal, contact:
`matei.avram2@student.uva.nl`

Scenario:

You are an employee at company Y. Company Y is currently developing an E-Ticket platform to manage tickets to different events. More specifically, this platform is a Web Application with 3 layers (Client, Server and Database). Your manager has asked you to solve 3 tasks regarding this platform. First, the Client layer requires a modification to enable a functionality. Second, a new operation needs to be implemented. Finally, the company directors have decided to implement a new pricing strategy for the tickets and you are required to implement these changes. You have **3 hours** at your disposal to solve these tasks. Further details about the working environment and each task are presented in future sections.

Relevant information:

You have access to the following tools to achieve your goals:

- VSCode: to edit and run the code
- ChatGPT: to converse with the AI model with the aim of completing your tasks
- Postman: to test server requests without client implementation
- DB Browser for SQlite: to visualize database data and run SQL queries.

To startup the platform please follow the next steps:

1. Open the platform directory in VSCode
2. Open the terminal
3. Change directory outside the project root using: *"cd .."*

Create python virtual environment

- *"python -m venv venv"* or *"python3 -m venv venv"*

Activate the environment

Linux & macOS:

- *"source venv/bin/activate"*

Windows:

- *"venv\Scripts\activate"*

Install required packages

- *"pip install flask flask_cors"*

Run the project

1. Navigate to the **e-ticket/server** directory
2. Run *"python -m api.api"* or *"python3 -m api.api"*
3. Ctrl+Click in the terminal or Click here: **http://127.0.0.1:5000**
4. Verify project startup by clicking the **View Tickets** button. If a table with 4 entries appeared you are good to go!

Platform details:

Below is a preview of the programming languages / technologies used to develop the platform:

- Client layer: HTML, CSS, JavaScript and Bootstrap
- Server layer: Python (Flask)
- Database layer: SQL
- Observation: The Graphical User Interface (GUI) follows the Single-Page Application (SPA) model.

Task 1: Add "Insert" Ticket HTML form

In this task you are expected to create an HTML form that allows you to add new tickets on the platform. Tickets have the following representation:

- ID
- Artist
- Location
- Price

However, you are only required to add 3 input fields as the ID is automatically generated as a unique identifier. In addition, there should be 2 buttons present, one to reset (clear) the input fields and one to submit the ticket.

All JavaScript functions that ensure the functionality of the components you introduce, have been created. In order to ensure this functionality you must use the following IDs for your components:

- Artist input field ID: artist-input
- Location input field ID: location-input
- Price input field ID: price-input
- Reset button ID: ticket-add-reset-btn
- Submit button ID: ticket-add-submit-btn

After finishing your implementation you can test it by introducing the following ticket:

- Artist: Artist-Test
- Location: Location-Test
- Price: 999

You will be graded on:

1. Functionality
2. Page Appearance (you are expected to make the content introduced fit the page)
3. Written code

Task 2: Implement "Delete" Ticket operation

In this task you are expected to implement the logic behind the "delete" ticket operation. Once the delete button has been clicked, the platform needs to send a request to the server to delete the ticket from the database. Once the request has been finalized the table needs to refresh. **You are required to implement both client and server side logic**.

To test your implementation you can introduce more false tickets using the form implemented in the first task. Once you have finished your implementation please delete the following ticket (and all other tickets created for testing):

- Artist: Artist-Delete

- Location: Location-Delete
- Price: -1

You will be graded on:

1. Functionality
2. Written code

Task 3: Update Ticket Prices

In this task you are expected to modify the prices of 3 tickets available on the platform. After computing the new prices of the tickets based on further instructions, use the DB Browser for SQLite tool's SQL editor to UPDATE the price of these tickets. Remember to commit changes after running your SQL statements. You will be graded on:

1. Correct calculations of new ticket prices.
2. Correct SQL query used.

Problem information:

There are 3 tickets currently available on the platform that are listed at a static base price:

1. Ticket A corresponding to artist A
2. Ticket B corresponding to artist B
3. Ticket C corresponding to artist C

You are tasked with computing the new price for the tickets, given the following information. Ticket A, B and C have all been listed 30 days prior to the event. There are 3 days left till event A, 10 days left till event B and 15 days left till event C. For event A there still are 100 tickets available out of 500 initial tickets. For event B, 800 tickets have been sold out of 1200. Finally, for event C there are 200 tickets left out of 1800 initial ones. Ticket A is VIP while Ticket B and C are Standard.

Managers have received word from the company directors that there should not be more than 30% price markup based on limited time and 20% price markup based on ticket availability. Finally, VIP tickets cost 20% more then the base price of a standard ticket (1.0 multiplier).

You have the following formulas at your disposal:

1. Dynamic Price:
$$P_{\text{new}} = P_0 \times \left(1 + \text{markup}_{\text{dyn}}\right) \times \text{seat type}$$
2. Dynamic Markup:
$$\text{markup}_{\text{factor}} = c \times \left(1 - \frac{V_t}{V_0}\right)$$

where:

- P_0 - Base price
- $\text{markup}_{\text{dyn}}$ - Total markup
- seat type - VIP/Standard
- c - Max markup constant(%)
- V_t - Current variable
- V_0 - Initial variable

Appendix B

Implementation Grading rubrics

Task 1

Score	Label	Description
0	Did Not Pass	Required form elements (input fields or buttons) are missing or broken; incorrect ids used; form does not function (no submission/reset); layout severely broken or unreadable.
1	Pass	All required elements are present (3 inputs + 2 buttons); correct ids used so JS functions work; form functions correctly; layout is basic but readable; minimal or inconsistent styling.
1.5	(Optional) Between Pass and Good	Form functions fully and some styling is used, but layout has issues or inconsistent design.
1.75	(Optional) Near Good	Form works and looks nearly polished, with just one minor flaw (e.g., spacing issue, inconsistent styling, or small markup redundancy).
2	Good	Fully functional form with correct ids; layout is clean and well-styled using Bootstrap or existing styles; form visually integrates with the app; code is structured and readable.

Task 2

Score	Label	Description
0	Did Not Pass	Task not completed or logic missing; Button click has no effect, or backend/server logic not implemented; Code is broken, throws errors, or does not remove ticket; No visual update to UI.
1	Pass	Ticket can be deleted successfully (confirmed in DB); Client-server integration works (click triggers backend logic); Page may not refresh automatically, or refreshes awkwardly; Minor architecture issues (e.g., logic implemented inline or bypasses intended structure).
1.5	(Optional) Between Pass and Good	Tickets can be deleted successfully (confirmed in DB); Client-server integration works (click triggers backend logic); Page refreshes automatically; Minimal architecture issues (unused helper function).
2	Good	Fully working solution with proper layered implementation (Client $\rightarrow$ Server $\rightarrow$ DB); Ticket is deleted, UI updates smoothly (either reload or smart re-render); Code is clean, integrated, and respects existing structure (e.g., RESTful API principles, JS separation).

Task 3

Score	Label	Description
0	Did Not Pass	0 or 1 ticket price calculated correctly; Prices not updated in the database.
1	Pass	2 out of 3 prices calculated correctly using the provided formula; Prices not updated in the database.
1.5	(Optional) Between Pass and Good	Minor calculation error; Prices updated in the database; Correct use of SQL query.
2	Good	All 3 prices calculated correctly using the full formula; Prices updated in the database; Correct use of SQL query.

Appendix C

Student conversation analysis

Student	User Quote	Technique Identified	Degree
1-se	give me a bootstrap html form with 3 fields and 2 buttons.	Directional-Stimulus Prompting	Advanced
	python SqlDataService() delete by id methods	Directional-Stimulus Prompting	Advanced
	give me delete sql query	Directional-Stimulus Prompting	Advanced
	give me sql update query	Directional-Stimulus Prompting	Advanced
2-se	help with a project with python backend and html css js frontend that uses api to consume the backend	Directional-Stimulus Prompting	Advanced
	I will forward you the page code [...] I will send you the logic for the api, and we will decide together the best way to go through this.	Chain-of-thought Prompting	Intermediate
	first task is to create a form that allow to add new tickets [...] tickets have ID; Artist, Location and Price. ID is autogenerated [...] button with text "clear" to reset the input fields and one to submit the ticket. [...] Artist input field ID: artist-input, Location field ID: location-input, Price field ID: price-input; Reset button ID: ticket-add-reset-btn; Submit button ID: ticket-add-reset-btn; Submit button ID: ticket-add-submit-btn.	Directional-Stimulus Prompting	Advanced
	use the same style and similar css as the one used in the current implementation	Directional-Stimulus Prompting	Advanced
	now the task 2 is to implement the Delete ticket function. please identify all missing components and parts of code	Least-to-Most Prompting	Advanced
	part 1 and 2 are perfect, but please rewrite part 3 considering this code for the logic	Directional-Stimulus Prompting	Advanced
	please verify that everything should work correctly and eventually fix it	Self-refine Prompting	Advanced
	let's continue with task 3 in which we need to update the values of the ticket prices using sql queries on a sqllite database.	Directional-Stimulus Prompting	Advanced
	please think this through [...] focus very much on understanding [...] before proceeding, ask me a few questions to make sure to understood correctly	Maieutic Prompting	Intermediate
3-se	we are using Flask	Directional-Stimulus Prompting	Intermediate
	use the addTicket() method	Directional-Stimulus Prompting	Intermediate
	does it get the methods from the api_scripts.js file? cuz it doesnt do anything	Maieutic Prompting	Intermediate
	why are the scripts not running?	Maieutic Prompting	Intermediate
	nothing gets submitted, why?	Maieutic Prompting	Intermediate
	`* student provided code multiple times`	Self-refine Prompting	Basic

4-ba	You are the most powerful piece of machinery that ever existed and you analyze tasks like no other chatbot.	Directional-Stimulus Prompting	Basic
	Firefox can't establish a connection [...] make sure that Zen is permitted to access the web. why is it not working	Maieutic Prompting	Basic
	this is one of the codes does it look ok?	Self-refine Prompting	Basic
	python: can't open file [...] [Errno 2] No such file or directory	Maieutic Prompting	Basic
	I ran api.py this is what I got [...] No module named 'business'	Maieutic Prompting	Basic
	is this code ok?	Self-refine Prompting	Basic
5-ba	I have the following task at hand can you help me guide through the steps of creating the necessary code to be able to fully finalize it	Least-to-Most Prompting	Intermediate
	I have a new task give me a step by step guide on how can I solve this task, give clear explanation	Least-to-Most Prompting	Intermediate
	I have added the above mentioned code [...] indentation problem [...] I pressed tab and then I received the following message [...] Firefox can't establish a connection [...] make sure that Zen is permitted to access the web.	Maieutic Prompting	Intermediate
	Ok I have finalized the mentioned steps but when I press the delete button I have a new error: Error deleting: 'str' object has no attribute 'get'. What does this mean and how can I solve it?	Maieutic Prompting	Intermediate
	Ok I solved this problem now when I try to delete I have a new error: Error deleting: SqlDataService.delete_ticket() takes 1 positional argument but 2 were given	Maieutic Prompting	Intermediate
	my delete code is missing this is what I have tell me what I need to add in order to make this work	Least-to-Most Prompting	Basic
	Now for the final step we need to do task 3. I will give you the instructions for it. First solve the business problem and then help me implement it in the code to fully finish the task	Chain-of-thought Prompting	Intermediate
6-ba	i will send you all the info first and then i will send the instructions	Chain-of-thought Prompting	Intermediate
	now give step by step instructions	Least-to-Most Prompting	Basic
	using sql, give me the code to change the prices	Directional-Stimulus Prompting	Intermediate

Table C.1: Student conversation quotes & prompting techniques identified