HUMAN DEVELOPER VS AI: A COMPARATIVE ANALYSIS OF MODERN AI BASED SOFTWARE DEVELOPMENT TOOLS

Raja Arslan Khan

01-241222-008

A thesis submitted in fulfillment of the
requirements for the award of the degree of
Master of Science (Software Engineering)

Department of Software Engineering

BAHRIA UNIVERSITY ISLAMABAD

November 2024

APPROVAL FOR EXAMINATION

Scholar's Name: Raja Arslan Khan, Registration No: 01-241222-008
Program of Study: MS (Software Engineering)
Thesis Title: Human Developer vs AI: A Comparative Analysis of Modern AI based Software Development Tools

This is to certify that the above scholar's thesis has been completed to my satisfaction and, to my belief, its standard is appropriate for submission for examination. I have also conducted a plagiarism test of this thesis using HEC-prescribed software and found a similarity index 7% that is within the permissible limit set by the HEC for the MS degree thesis. I have also found the thesis in a format recognized by the BU for the MS thesis.

Principal Supervisor's Signature: ________________________
Date: 13/11/2024_
Name: Dr. Awais Majeed

AUTHOR'S DECLARATION

I, Raja Arslan Khan hereby state that my MS thesis titled "Human Developer vs AI: A Comparative Analysis of Modern AI based Software Development Tools" is my work and has not been submitted previously by me for taking any degree from this university Bahria University Islamabad or anywhere else in the country/world.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw/cancel my MS degree.

Name of scholar: Raja Arslan Khan (01-241222-008)

PLAGIARISM UNDERTAKING

I, Raja Arslan Khan, solemnly declare that the research work presented in the thesis titled "Human Developer vs AI: A Comparative Analysis of Modern AI based Software Development Tools" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero-tolerance policy of the HEC and Bahria University towards plagiarism. Therefore, I as an Author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred to/cited.

I undertake that if I am found guilty of any formal plagiarism in the above-titled thesis even after the award of my MS degree, the university reserves the right to withdraw/revoke my MS degree and that HEC and the University have the right to publish my name on the HEC/University website on which names of scholars are placed who submitted plagiarized thesis.

Scholar / Author's Sign: ________________________

Name of the Scholar: Raja Arslan Khan (01-241222-008)

DEDICATION

To my beloved mother and father

ACKNOWLEDGEMENT

With my deepest gratitude to Allah Almighty, I would like to start this acknowledgment. I extend my appreciation to my thesis supervisor, Dr. Awais Majeed for his valuable feedback and mentorship in shaping this work. I would like to thank my parents for their support and belief in me, are the greatest source of motivation for me.

ABSTRACT

Use of AI based tools have gained extensive acceptance by the software development community in the latest past. Various tools have now become an integral part of IDEs. At the same time AI based software development is also emerging with the advancement in generative AI. EngineerGPT, GitHub Copilot and ChatDev are a few examples of such applications. However, effectiveness of these tools still needs to be evaluated throughout the Software Development Life cycle phases helping in developing end-to-end applications. To make comparison about human developer efficiency and AI tools, criteria of feature completeness, quality of code and test comprehensiveness was developed. Two software applications having standard software specifications were given to human developers as well as AI based tool (EngineerGPT). The output generated by both was later compared based on the above-mentioned criteria. For Code quality we have used the indicators of Cyclomatic Complexity, Lines of Code (LOC), and Code Duplication. The results of the output reveal that while AI-driven tools efficiently implement core functionality with compact codebases, low duplication, they demonstrate limitations in handling complex requirements and modular design, which can affect code adaptability and feature alignment. In contrast, human developers produce more verbose and modular code, utilizing frameworks and libraries to enhance maintainability. Code based of human developers was large with duplicated code. Test case analysis further highlights differences in coverage, with human-driven approaches achieving complete validation across all requirements, while AI-driven tools effectively cover primary functions but lack thoroughness in secondary features. This research demonstrates the strengths and limitations of AI in software engineering, indicating that while AI-driven tools hold potential for rapid, core functionality development, human expertise remains critical for achieving robust, maintainable, and fully compliant software solutions. The insights contribute to a deeper understanding of AI's role in software engineering, with implications for optimizing human-AI collaboration in future development workflows.

Table of Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

The integration of artificial intelligence (AI) in software engineering (SE) has grown significantly. Recent studies, have highlighted AI's ability to streamline tasks like requirements gathering, code generation, and testing, indicating its transformative potential within SE. As AI technologies advance, their ability to assist or even partially replace human roles in software development tasks has sparked interest of the community AI's utilization in specific areas to get most value [1], [2].

However, AI's role in SE is not without its challenges. Although, AI tools like ChatGPT and Copilot bring creativity and efficiency to SE, they also introduce technical challenges. These include data dependency, solution reliability, and the need for carefully managed prompts to generate consistent and accurate outputs [3], [4]. Systematic reviews, such as Feldt et al. [5] and Nguyen-Duc et al. [6], have categorized these AI applications along with the insights about their potential benefits and the technical and ethical hurdles that needs to be addressed for completely integrating AI into Software Development processes.

Comparative analysis of AI-driven tools versus human developers have provided further insights into the strengths and limitations of both approaches. Studies like Nascimento et al. [7][8] demonstrate that AI can enhance efficiency in functional and non-functional SE tasks, more efficient with human expertise being essential for tasks requiring complex understanding and adaptability. Imai [9] and Ahmad et al. [10] discuss how human developers and AI together play a co-operative role and what does this imply for AI's potential ability to enhance rather than replace humans into software engineering, especially in complex or high-stakes projects.

Sauvola et al. [11] and France et al. [12] discussed future scenarios and ethical considerations, highlighting that while AI promises to enhance productivity, it also raises concerns regarding intellectual property, job displacement, and the responsible use of AI in SE. This diversified challenge emphasizes continued research into the role of AI in Software Engineering and balancing gains in productivity with ethical responsibilities.

This research aims to contribute to this understanding by conducting a comparative analysis of AI-driven versus human-centric software development. Using key metrics like quality of code, features alignment to specification, and test case coverage, this paper will provide insights like to what extent AI can be integrated into conventional SE practices, considering its strengths and limitations. Through this analysis, the aim of this research is to provide a comprehensive view of AI's role and future direction in software engineering.

1.1. Background

With the evolution of artificial intelligence (AI), it is now deeply integrated in the software engineering process from requirements to deployment. Studies have highlighted AI's role in automating routine tasks, analyzing large datasets, and optimizing decision-making. This systematic integration enables AI-driven models to assist in tasks like code generation, testing, and maintenance, demonstrating the technology's potential to streamline SE workflows [3], [2], and [13]. In addition, application of various AI based applications in software engineering based on their functional role, AI technology type, and level of automation, have also been suggested for effective deployment of AI in various activities in software engineering [5].

A growing body of research has focused on comparing the efficacy of AI models with that of human developers. Findings reveal the collaborative potential of human and AI capabilities, with each excelling in different aspects of software engineering activities[7][8].Some studies suggest that while AI can greatly assist SE tasks, human expertise remains critical for comprehensive project success especially in ensuring the quality of code generated [9].

Although AI offers significant advancements, its integration into software development is not without challenges. Studies highlight that, while AI-driven tools improve efficiency and creativity, but there are some technical hurdles such as data dependency, prompt accuracy, and maintaining solution quality [4],[14],[15],[16]. Additionally, Bird et al. [17] and Pudari et al. [18] noted that tools like Copilot occasionally miss coding best practices, highlighting the importance of reliable prompts and effective human oversight. These challenges underline the need for hybrid approaches that allow integrating traditional SE practices with AI-supported capabilities which maximize productivity, quality of the code, and produce the best code possible.

The rapid advancements of AI prompted researchers to consider its future within Software Engineering. One of the challenge is the potential threat of job loss for software engineers especially developers with AI based automation tools in various SE activities [11]. In Addition, there are some concerns about intellectual property, security, and transparency that have also been raised by the industry. [12]. Wang et al. [19] have also surveyed early-stage research on LLM-based autonomous agents and found some gaps in research for the practical application of such models. Together, these studies recommend strategic planning and ethical guidelines with AI becoming a substantial force in SE.

In summary, AI in software engineering has a dynamic background of innovation and comparative insight into human expertise alongside the continuing challenges and future considerations. This foundation highlights the importance of continued research and development in optimizing the role of AI to be associated with productivity and ethics as the field evolves.

1.2. Research Gap

AI-driven tools like EngineerGPT and GitHub Copilot have brought advancements to software engineering, automating many tasks and boosting productivity. However, a crucial research gap remains when it comes to fully understanding how these tools stack up against traditional human-driven development. While studies have shown that AI-assisted code generation (e.g., with GitHub Copilot) can increase efficiency, they often highlight a trade-off in the quality of the generated code, particularly in pair programming scenarios [9]. Despite these findings, there has been little to no focused research directly comparing software developed entirely by AI tools to that created by human developers across key aspects such as feature alignment with specifications, code quality, and test coverage.

The key areas where this gap is evident include:

Feature Alignment with Specifications: Research is very limited for evaluating how well AI versus human developers interpret and implement features from the same set of specifications.

Code Quality Comparison: Limited studies exist that directly compare the code quality of AI-generated software against human-developed software from identical specifications and designs.

Test Case Development: There's a notable lack of comparative analysis to assess how comprehensive and effective test cases developed through AI tools are compared to those developed by human developers, using the same specifications and designs.

Addressing these research gaps will be very important to uncover implications, challenges, and opportunities presented using AI-driven software development tools such as EngineerGPT. This analysis will be valuable for optimizing AI-human developer collaboration in efficient and innovative software engineering practices.

1.3. Problem Statement

The introduction of AI-based software development tools such as EngineerGPT went on to change the scenario of software engineering with newer possibilities offered for automating the development process. However, the effectiveness of these AI based tools versus human driven methods still needs to be established in terms of quality of code generated, conformance to given specifications and development of test cases. This study aims to address this gap by systematically comparing the outputs of AI-driven and human-developed software projects from identical specifications and designs, to find out the reliability, efficiency, and potential of AI in enhancing software development practices.

1.4. Research Questions

To address the identified problem and achieve the research objectives, the following research questions (RQs) have been formulated to guide the investigation into the comparative analysis of AI-driven tools like EngineerGPT and traditional human developers in software engineering:

RQ1. To what extent do AI-generated software and human-developed software align with the provided specifications in terms of feature implementation?

RQ2. How does the code quality of software developed by AI-driven tools like EngineerGPT compared to that developed by human developers when given the same specifications and design?

RQ3. How do test cases developed by AI tools compare in comprehensiveness and effectiveness to those created by human developers, following identical specifications and designs?

Answering these research questions will be crucial to provide a clear understanding of the strengths, limitations, and broader implications of using AI-driven tools in software engineering. It is important to understand these strengths and limitations for optimizing the use of AI in the field along with guiding future developments and best practices.

1.5. Research Objectives

Considering the research gaps in the comparative analysis of AI-driven software development using tools like EngineerGPT versus traditional human-driven methods, to guide this research the following objectives are established:

Assess Feature Implementation: This evaluates how correctly features of AI-generated software and human-developed software reflect the features described in the specifications, measuring the completeness and functionality of the features developed.

Evaluate Code Quality: To compare the quality of code generated by AI-driven tools like EngineerGPT with human developers given the same specifications and design, maintaining quality dimensions related to maintainability, efficiency, and best practice.

Compare Test Case Development: To compare test cases developed by the AI tools with human-developed test cases, based on their ability to identify critical functionalities covered in the developed software.

Through these objectives, the goal is to provide a comprehensive understanding of the capabilities, limitations, and broader implications of AI-driven software development tools compared to traditional human-driven methods. This understanding is crucial for determining optimal strategies for integrating AI in software development and for guiding future innovations and best practices in the field.

Chapter 2

LITERATURE REVIEW

The integration of AI tools in software engineering tasks has become a significant area of interest in the recent decades due to rapid development in the fields of AI especially Large Language Models (LLMs) and transformer models. Increasingly many existing software engineering, and software development tools have integrated AI assistance in one way or other the enhance performance and efficiency of software engineers and developers. In this chapter we will explore the existing state of the art in AI based Software Development to explore the capabilities, limitations, and implications of AI based tools especially the Generative AI based tools in the field. This literature review synthesizes findings from various studies, providing insights into the current state of AI-augmented software engineering.

2.1. AI Tools and Techniques in Software Engineering Phases

Sofian *et al.* [1] conducted a systematic mapping study to categorize AI applications across various phases of software engineering, identifying eight distinct SE phases where AI techniques are applied, with a notable concentration in requirements engineering, system development, and testing. Similarly, Barenkamp *et al.* [20] explored the role of AI during the entire software development lifecycle by combining a systematic review with developer interviews. The study highlighted achievements of AI and its limitations within SE, especially its need for human oversight and human creativity to drive innovation. Batarseh *et al.* [21] further elaborated on the impact of AI by discussing its core paradigms, including neural networks, machine learning, and natural language processing, which they proposed as a framework for advancing AI in SE practices. This further categorization was introduced with the AI-SEAL taxonomy by Feldt *et al.* [5], a structured approach that classifies AI applications based on their functional points within SE, along with the types of the AI technologies used, and their automation levels which offers a strategic tool for organizations adopting AI in software processes. Finally, Nguyen-Duc *et al.* [6] reviewed and conducted a focus group study of Generative AI in SE to identify critical research gaps and development needs to increase the utility of AI in real-world SE applications. Together, these studies establish a basis for understanding how AI is evolving and changing its role in software engineering processes by both providing the potential and showing the challenges of making AI a significant part of the SE workflow process.

2.2. Performance and Capabilities of Specific AI Tools in SE

Bird *et al.* [17] explored the effects of Copilot on software development and exemplified on the Codex-trained version of a large data from GitHub which would direct the developer's activities from mostly executing duties to more evaluation-based ones. Russo *et al.* [4] provided further information on adopting LLM in SE through structured interviews, which pointed out some of the benefits such as improved efficiency and code quality, alongside some limitations in user experience.

In a broader analysis, Hou *et al.* [3] conducted a systematic review on 229 papers on LLM applications in SE. The researchers categorized the types of LLMs, strategies in data management, and methods of evaluation of performance to identify focus areas where LLMs are more prevalent in SE tasks. The review highlighted trends, identified research gaps, and outlined future directions for improving application of LLM in SE. In a practical evaluation, Imai *et al.* [9] assessed GitHub Copilot's effectiveness compared to traditional human pair programming. The study found that although there was an increase in productivity, this came at the potential cost of decreased code quality, which calls for a balance between the improved workflow and reduced quality. Collectively, existing work provides an insight into the strengths, the limitations and evolving role of AI tools within software engineering.

2.3. AI and Human Comparisons in Software Development

Nascimento *et al.* [7][8] compared human and AI performance in software engineering, also engaging with both functional and nonfunctional requirements. Their findings indicated potential for effective collaboration between human developers and AI across a variety of SE tasks along with bringing unique strengths to the development process. In a forward-looking study, Shastri *et al.* [22] interviewed technology professionals to assess the changing impact of AI on the software development lifecycle with the conclusion that, in the next few years or so, AI can be relied upon for handling production-quality code, thereby potentially reshaping the developer's role in SE.

Fan *et al.* [14] took a different approach by focusing on hybrid AI techniques in SE, which combine traditional software engineering practices with large language models (LLMs). This integration was shown to foster creativity and enable AI to tackle complex problems, though challenges such as hallucinations and solution accuracy persist. In total, these research show some promising application fields of AI in software engineering, mainly if human expertise is combined with AI-driven tools.

2.4. AI-Assisted Programming and Productivity

Ernst and Bavota [13] discussed the role of AI-Driven Development Environments (AIDEs) in automating routine tasks within Integrated Development Environments (IDEs). Their study highlighted Codex's potential to streamline software engineering processes but brings out concerns about bias, copyright, and the future of programming education at large. Ozkaya *et al.* [23] also performed a methodological survey of LLM applications in SE, noticing usability for increases in productivity in terms of generating some tasks, such as code generation or language translation. Even though they have identified challenges to be overcome for the efficient LLM integration, like the quality of data and misinformation.

France *et al.* [12] analyzed the productivity impact of ChatGPT and GitHub Copilot by examining social media discussions and survey data. While these tools were found to enhance productivity but there are some concerns that emerged around intellectual property and security risks. Moving forward, Sauvola *et al.* [11] proposed scenarios for AI integration within SE, recognizing productivity gains but also highlighting ethical concerns such as job displacement and the need for responsible AI practices. In summary, these studies indicate that AI could offer potential benefits in terms of productivity improvement and point to some of the challenges in the ethical and practical practice of SE as brought about by AI.

2.5. AI-Generated Code Quality and Code Review

Mastropaolo *et al.* [15] evaluated GitHub Copilot’s robustness in code generation while also observing that the quality of generated code is highly sensitive to input descriptions which directly affects its accuracy. Going further, Korada *et al.* [16] reviewed Copilot's influence on the software application lifecycle, they report increased productivity but again caution about the dependency on the specificity of the prompt and still pending issues of Intellectual Property (IP). Similarly, Pudari *et al.* [18] analyses the extent to which Copilot follows standard coding practices and while Copilot often provides syntactically correct code, at times it misses idiomatic conventions and optimized solutions.

Dakhel *et al.* [24] conducted a comparative study between Copilot-generated code and human solutions. The study concludes that while AI-generated errors are often easier to fix, complex tasks still require human oversight. Adding to these findings, Zhang *et al.* [25] explored Copilot’s usage across various programming environments. This reports that while it can accelerate development, integration and code quality issues arise in more complex tasks. These studies highlight the benefits and constraints of using GitHub Copilot, especially concerning complex tasks such as understanding, best practice compliance, and stable and reliable integration with various coding environments.

2.6. Evaluation Metrics in Software Engineering

To assess and compare the quality of AI-driven and human-developed software, specific code metrics are required. These metrics provide quantitative insights into code complexity, maintainability, and optimization. It enables a systematic evaluation of AI-generated versus human-generated code. Key metrics: Cyclomatic Complexity, Lines of Code (LOC), and Code Duplication are widely recognized for their relevance in analyzing code quality in software engineering.

2.6.1. Cyclomatic Complexity

Cyclomatic Complexity is a measure of the number of linearly independent paths through a program showing the code complexity. This metric highlights the potential challenges in maintainability and readability of the code. The higher complexity indicating more complex logic and decision points that can hinder code understanding and increase the likelihood of errors.

Dakhel *et al.* [24] used Cyclomatic Complexity to evaluate the readability and understandability of code produced by GitHub Copilot for comparing it to human-generated solutions.

2.6.2. Lines of Code (LOC)

Lines of Code (LOC) provides a quantitative measure of codebase size for offering insights into productivity, verbosity, and code simplicity. While a higher LOC count may indicate more comprehensive functionality, the lower LOC can depict a more concise and efficient solution with better readability and quality are maintained.

Imai [9] utilized LOC to evaluate productivity when comparing Copilot assisted programming to traditional human pair programming.

2.6.3. Code Duplication

Code Duplication assesses the presence of redundant code segments within a codebase, which can indicate missed opportunities for refactoring and optimization. High levels of code duplication suggest that similar functionalities are repeated unnecessarily, increasing maintenance costs and the potential for inconsistencies.

This reflects the different ways of Software Quality: Cyclomatic Complexity, Lines of Code, and Code Duplication. Each one gives a precise measuring to compare the maintainability, productivity, and optimization of the code in comparison between AI-driven and human developed code. Backed by previous literature in support of these metrics, it would allow for a more comprehensive strength and weakness analysis between both approaches and how an AI influences effective software engineering.

2.7. Future Directions and Potential of AI in SE

Wang *et al.* [19] have surveyed over LLM-backed autonomous agents, where these promise early applications but give an immense research gap in enhancing them for practical and reliable usability. Similarly, Dong *et al.* [26] investigated collaborative role-based code generation using multiple LLM agents. Providing a finding that performance can be enhanced through agent collaboration. However, the study also noted challenges with scaling these techniques to handle the complexity of real-world applications.

Latinovic *et al.* [27] approached the topic of AI and automation from a practitioner's perspective, conducting interviews that revealed current automation in software engineering tends to be task-specific, providing limited transformative impact on SE processes overall. Aligning with this perspective, Ozkaya *et al.* [28] advocated for AI-driven automation in SE, particularly using bots to handle repetitive tasks, thereby enhancing productivity. However, the authors emphasized the need for practical case studies and examples to demonstrate these bots' effectiveness in varied SE environments. These studies collectively underscore the potential for AI and automation in SE while noting the challenges of real-world implementation and the need for further practical research.

Table 1 summarizes the state of the art in the domain of AI driven software development.

Table 1: Comparative Analysis of existing state of the art

Sr #	Paper	Proposed Approach/Methodology	Findings	Limitations
1	Sofian et al. [1]	Systematic mapping study characterizing AI applications in SE, with inclusion/exclusion criteria for paper selection and results analysis.	Mapped AI techniques across SE phases, identifying research needs in AI applications for SE.	Potential omissions due to selection criteria, lacks in-depth analysis of AI techniques' effectiveness.
2	Ozkaya et al. [2]	Reviewed AI's role in SE evolution, with focus on Agile, DevOps, and "test-first" methods.	Highlighted efficiency benefits of AI-based tools in improving SE processes, especially in DevOps.	Lacks specific details on tools and challenges of AI adoption in SE.
3	Hou et al. [3]	Systematic literature review analyzing 229 papers from 2017-2023 on LLMs in SE, categorized by tasks, data handling methods, and evaluation strategies.	Provided comprehensive understanding of LLM applications, gaps, and trends in SE. Highlighted promising areas for future study.	Dependent on paper quality; selection criteria may introduce bias. Limited to papers from 2017-2023, possibly excluding relevant earlier research.
4	Russo et al. [4]	Mixed-method approach with structured interviews and analysis using theories like TAM and DOI. Developed HACAF framework for organizational strategies, validated with PLS-SEM data from 183 professionals.	Insights on LLM adoption in SE, showing that LLMs enhance time efficiency, code quality, and user experience.	Sample may not represent the SE population fully; limited by lack of follow-up inquiries affecting data depth.
5	Feldt et al. [5]	Applied the AI-SEAL taxonomy to classify AI use in SE, examining 15 papers from RAISE workshops.	Demonstrated AI-SEAL's utility in classifying AI applications and identifying associated risks.	Focused mainly on classification without in-depth risk analysis of each AI application.
6	Nguyen et al. [6]	Literature review and focus groups over five months, developing a research agenda with 78 open questions in 11 SE areas.	Provided a comprehensive understanding of GenAI's potential and limitations in SE, suggesting directions for future research.	Limited studies in real-world environments, with potential biases in generated requirements and fairness issues.
7	Nascimento et al. [7]	Empirical comparison using LeetCode problems, generating code with ChatGPT and assessing	Found that ChatGPT-4 solutions outperformed existing ones in performance and	Limited scope in comparing AI vs. human performance; training and testing

		solutions for performance and efficiency.	memory efficiency on certain tasks.	data may affect validity.
8	Nascimento et al. [8]	Evaluated ChatGPT's capabilities in SE tasks across syntax, static behavior, and dynamic behavior using tasks in C, Java, Python, and Solidity.	ChatGPT showed initial competence in static code analysis, similar to AST parsing.	Limited by absence of large datasets and automatic prompts, constrained by token limits and model's capacity to handle complex dynamic semantics.
9	Imai et al. [9]	Experiment with 21 participants coding in three scenarios: with Copilot, as driver, and as navigator, with random assignments.	Copilot boosted productivity but resulted in slightly lower code quality, assessed by lines added and deleted.	Limited to productivity and quality measurements; small sample size without considering experience levels.
10	Ahmad et al. [10]	Case study using ChatGPT to assist a novice architect with collaborative human-bot architecting.	ChatGPT assists in generating architectural decisions and models.	Requires iterative refinements due to varied responses; limited generalizability to complex systems.
11	Sauvola et al. [11]	Analytical approach with four scenarios predicting generative AI's impact on software development.	Highlights productivity boost in repetitive tasks, potentially lowering costs.	Ethical concerns around IP, job displacement; theoretical scenarios lacking empirical data.
12	France et al. [12]	Qualitative analysis of developer discussions on Reddit, combined with a literature review and adoption survey.	Provided timely insights into AI tools' impact on SE, showing increased efficiency in routine coding tasks.	Heavy reliance on social media, potentially unrepresentative of the wider SE community.
13	Ernst et al. [13]	Discussed challenges and benefits of AI-driven development in IDEs, using Codex and software repository mining.	Codex showed promise in automating routine tasks but raised concerns about bias and legal issues.	Potential for copyright conflicts and dependency on model quality, with limited control over code generation.
14	Fan et al. [14]	Survey on LLMs in SE, identifying challenges and the role of hybrid techniques that combine traditional SE and LLMs.	Found LLMs support creativity in coding, design, and requirements engineering, aiding	Technical challenges remain in eliminating hallucinations and incorrect solutions; further exploration

			documentation and classification tasks.	needed for test generation.
15	Mastropaolo et al. [15]	Used Copilot for Java method generation based on Javadoc descriptions, testing changes in recommendations.	Found 46% variability in code recommendations, with 28% affecting correctness.	Focused only on Java methods; limited scope in languages and contexts beyond Java.
16	Korada et al. [16]	Review of GitHub Copilot's features, with content analysis of developer experiences and industry case studies.	Enhanced productivity, improved secure coding practices, and faster feedback loops.	Dependence on prompt accuracy, IP concerns, and potential security issues in AI-generated code.
17	Bird et al. [17]	Analysis of user experiences with GitHub Copilot on code suggestions and tasks.	Copilot effectively adapts to various styles and languages, enhancing code completion.	May lack defensive coding practices; findings rely on subjective feedback.
18	Pudari et al. [18]	Exploratory study analyzing Copilot's code suggestions for language idioms and code smells, introducing a taxonomy for AI-supported tools.	Copilot generates syntactically correct code and helps with basic functionality.	Struggles with idiomatic practices and avoiding code smells, limited by its training data for optimized code quality.
19	Wang et al. [19]	Comprehensive survey of studies on LLM-based autonomous agents, covering construction, applications, and evaluation.	Offered a unified framework for LLM-based agents, covering applications in SE, social science, and engineering.	Field still in early stages with limited research progress and detailed methodology results.
20	Barenkamp et al. [20]	Mixed-method approach with a systematic review of 60 studies and qualitative interviews with software developers.	Provided a comprehensive view of AI's opportunities and risks in SE.	Limited interviews (only five), and lacks evaluative tools for AI applications in SE.
21	Batarseh et al. [21]	Reviewed AI methods applied to SE from 1975 to 2017 across SE phases like requirements, design, development, testing, and maintenance.	Provides a comprehensive perspective on AI methods across SE, challenging traditional views.	Limited to methods up to 2017; excludes recent advancements and lacks in-depth effectiveness analysis.
22	Shastri et al. [22]	Empirical study with interviews and surveys on AI's effect across SDLC phases, using thematic analysis.	AI accelerates SDLC phases like planning, design, and testing.	High reliance on input accuracy; limited by small sample size and generalizability issues.

23	Ozkaya et al. [23]	Discussed LLM use in SE tasks like code generation, language translation, and documentation, emphasizing productivity and ethical challenges.	LLMs show productivity improvements in tasks like code generation, though trust issues arise from data quality concerns.	Limited by inability to trace recommendations to sources and lacks practical testing in SE contexts.
24	Dakhel et al. [24]	Empirical study comparing GitHub Copilot's performance on fundamental tasks with human-provided solutions.	Copilot provides correct solutions for basic tasks, with easily repairable code.	Produces buggy, non-reproducible code; limited capability in solving complex problems.
25	Zhang et al. [25]	Empirical study using Stack Overflow and GitHub Discussions to analyze Copilot usage in programming.	Faster development and useful code generation, but challenges with integration and code quality.	Limited to self-reported data from forums, may not reflect a broader developer population.
26	Dong et al. [26]	Empirical study using role-based code generation with LLM agents (analyst, coder, tester) for improved performance.	Performance gains of 29.9%-47.1% on benchmarks compared to individual agents.	Limited optimization in complex tasks and benchmarks may not reflect real-world scenarios.
27	Latinovic et al. [27]	Semi-structured interviews with SE practitioners on automation and AI use in SE tasks.	Revealed common micro-automation practices in SE, with potential for cognitive overhead in automation.	Limited sample size, focused mainly on SMEs in Austria and neighboring regions, reducing generalizability.
28	Ozkaya et al. [28]	Discussed AI and ML-based bots for automating repetitive SE tasks, emphasizing criteria like bounded decision space.	Presented a perspective on SE automation, highlighting potential for software bots to streamline tasks.	Lack of case studies or specific examples of bot use; limited discussion of challenges or limitations.

2.8. Conclusion

This chapter explored the current research on AI-driven tools in software engineering, shedding light on how these technologies are transforming various phases of the development process, from requirements gathering to automated code generation, testing, and maintenance. The literature illustrates that while AI can accelerate many tasks, it also faces notable challenges, such as inconsistencies in output, difficulties with complex requirements, and issues with data quality and ethical concerns.

Comparative studies reveal a consistent trend: AI tools are unmatched in speed and efficiency for simpler tasks, yet human developers remain essential for projects requiring complex decision-making, intricate control structures, and comprehensive feature coverage. Key metrics like Cyclomatic Complexity, Lines of Code (LOC), and Code Duplication further highlight where each approach excels and falls short in terms of code quality and maintainability.

In short, the literature points to a promising yet balanced future for AI in software development. AI tools hold significant potential to boost productivity but are not yet a substitute for the creativity and adaptability of human developers. This conclusion sets the stage for the next chapters, where we'll explore these insights in practice, directly comparing AI and human-driven software development outcomes. Ultimately, this chapter suggests that a combined approach leveraging the speed of AI and the depth of human expertise may offer the best path forward in modern software engineering.

Chapter 3

RESEARCH METHODOLOGY

3.1. Introduction

As AI continues to advance, its application is software engineering more specifically its effectiveness relative to human-driven development has become a significant area of interest. This study adopts a quantitative research design to conduct a structured, comparative analysis of human-driven versus AI-driven software development across several critical metrics, including feature completeness, code quality, and test case coverage. The methodology is organized into five distinct phases, each contributing to an objective and consistent framework for evaluating both approaches.

The first phase, outlined in Chapter 2, involved a literature review to identify existing research on human and AI-driven software development. This review highlighted a significant research gap in direct comparisons between human and AI-driven development processes, establishing the need for this study. The second phase focused on developing a standardized software specification document using ChatGPT, ensuring that both human and AI developers operated from the same set of requirements to enable fair comparison.

In the third phase, Tool Selection and Rationale, an AI tool was chosen to support the study's goals of minimal human intervention and high output quality. EngineerGPT was selected based on its capability for autonomous software generation, quality of output, and strong reputation within the developer community. However, comparison of the output generated by ChatDev has also been carried out. The fourth phase involved the development process, in which both human developers and EngineerGPT implemented solutions based on the same specifications, allowing for direct comparison of outputs.

Finally, the fifth phase established an Evaluation Process to assess each solution on quantitative metrics, including code quality, feature completeness, and test case coverage. This structured evaluation framework enabled an objective analysis of the strengths and limitations of human-driven and AI-driven development approaches within software engineering.

3.2. Research Design

This study adopts a quantitative, comparative research design to assess human-driven versus AI-driven software development. The methodology is structured into five phases to ensure a consistent, data-driven evaluation across key metrics like feature completeness, code quality, and test case coverage.

3.2.1. Overview of Phases and Steps

Phase 1: Literature Review and Identification of Research Gap

- Conduct a literature review to identify gaps in current research on human-driven versus AI-driven development, providing the foundational context for the study.

Phase 2: Software Specification Development

- Generation of a standardized software specification is significant to provide standardized input to both generative AI based tools and human developers. It is important to avoid any variation in the specification while experimenting with both groups. ChatGPT was used to create this specification to provide a unified project foundation, ensuring both human and AI developers work from the same requirements.

Phase 3: Tool Selection and Rationale

- Assess and select an AI tool aligned with the study’s goals of minimal human intervention and high output quality.
- Although various tools very considered including ChatDev, Github Copilot and EngineerGPT, EngineerGPT was selected based on its autonomous software generation capabilities, output quality, and positive reputation within the

developer community with more than 50k stars on GitHub that's almost double of ChatDev.

Phase 4: Development Process

- **Human-Driven Development**: Assign the specifications to human developers, who follow a traditional iterative process to produce modular, quality-checked solutions.
- **AI-Driven Development with EngineerGPT**: Implement the specifications using EngineerGPT, adapting a modular and iterative prompt-testing cycle to manage tool limitations.

Phase 5: Evaluation Process

- Employ a structured evaluation framework to analyze each solution's feature completeness, code quality, and test case coverage using tools such as SonarCloud [1]for static code analysis.

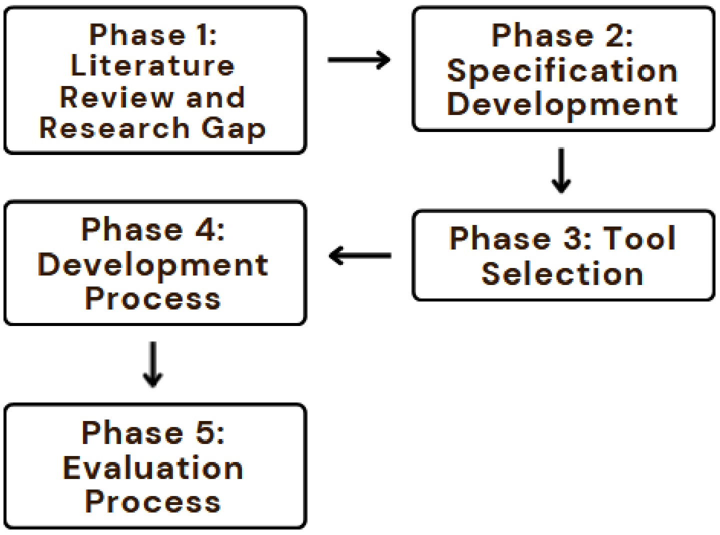

Phase 1: Literature Review and Research Gap
Phase 2: Specification Development
Phase 3: Tool Selection
Phase 4: Development Process
Phase 5: Evaluation Process

Figure 1: Overview of research methodology

Figure 1 summarizes the overall research design for the study conducted in this thesis. Each phase is further elaborated in the following sections.

[1] https://sonarcloud.io

3.2.2. Phase 1: Literature Review and Identification of Research Gap

As outlined in Chapter 2, a comprehensive literature review was conducted to evaluate existing research on human-driven and AI-driven software development approaches. This review identified a significant gap: the lack of direct, structured comparisons between the quality and completeness of outputs from human-driven versus AI-driven development processes. This gap informed the study's focus on measurable, side-by-side evaluations, guiding the methodology described in this chapter.

3.2.3. Phase 2: Software Specification Development

To enable a fair and consistent comparison of human-driven and AI-driven development, a standardized software specification document was created using ChatGPT. This document provided a unified set of requirements, which formed the basis for both human developers and EngineerGPT to work from the same guidelines. It was with an objective to ensure that differences in outputs reflected only differences in the development approach itself rather than variations in the interpretation of requirements.

The development of specification began with gathering the essential requirements for two distinct projects. CGPA & Transcript Application and the To-Do Application, which would offer a range of features and points for evaluation. ChatGPT was utilized to generate a comprehensive specification document detailing all required functionalities, user interactions, and performance expectations. This approach minimized human intervention in specification creation along with promoting the consistency and reducing the potential biases that could arise from manual adjustments.

The initial output generated by ChatGPT was reviewed and refined for the better clarity, coherence, and completeness. Minor modifications were made to remove the ambiguities and the document was carefully created to align with the evaluation criteria established for this study. This process of validation ensured that both development approaches, human-driven and AI-driven would go forward with an equal and a well-defined starting point. Consequently, the standardized specification document enabled an objective comparison of feature completeness, code quality, and test case coverage in the subsequent phases of the research.

3.2.4. Phase 3: Tool Selection and Rationale

The choice of tools used in this study was essential to ensure a fair, consistent, and robust comparison between human-driven and AI-driven development approaches. The goals of the study for each tool were held up against the criteria of minimizing human intervention, high-quality output, and the trust of tools in the developer community. Following these criteria, the selected tools were **EngineerGPT** for AI-driven development, **SonarCloud** for static code analysis along with e**xperienced human developers** for the human-driven benchmark.

EngineerGPT was chosen from some of its alternatives like ChatDev, ChatGPT, and GitHub Copilot due to its capability to autonomously generate full software applications with minimal human intervention. Unlike ChatGPT and GitHub Copilot, which produce only the code snippets that often require substantial manual integration and structuring. EngineerGPT is designed to independently create entire applications. This automated nature of the tool closely aligns with the study's objective of reducing human oversight in AI-driven development. Although ChatDev was initially considered, but the output of EngineerGPT was found to be more robust and self-contained which makes it a better candidate for direct comparison with human-driven development. The exclusion of ChatDev will be discussed in detail in the next chapter.

Initially based on the GPT-3.5 Turbo model, EngineerGPT was upgraded mid-study to the GPT-4o model, which provided enhanced capabilities for handling complex requirements and generating more cohesive, extensive outputs. A comparative assessment of the GPT-3.5 Turbo and GPT-4o models, using identical prompts, revealed that GPT-4o handled complex requirements more effectively, producing complete outputs with fewer interruptions. Furthermore, the popularity of EngineerGPT and its positive reception within the developer community demonstrated by its high GitHub rating and favorable reviews strengthened its credibility as a good choice for AI-driven development in this comparative study.

For the evaluation of code quality, SonarCloud was selected as the static analysis tool. It offers objective and standardized metrics essential to compare the AI-generated and human-developed solutions. SonarCloud provided crucial metrics for our study such as cyclomatic complexity, lines of code (LOC), and code duplication that allows for an in-depth assessment of each codebase's maintainability and complexity. The choice of SonarCloud supported a structured analysis of code quality with its consistent basis for evaluation that could be applied uniformly across all solutions.

Human developers were chosen based on their experience levels. Specifically targeting mid-senior software engineers to establish a realistic and competitive standard for the comparison. These developers were assigned the same specifications that were provided to EngineerGPT. Their contributions served as a benchmark for the study, allowing for a direct, structured comparison of human and AI-driven development capabilities.

3.2.5. Phase 4: Development Process

In this phase, both human and AI-driven development approaches were executed using the standardized software specifications created in Phase 2. By ensuring that each approach followed the same project requirements, this phase enabled an objective comparison of the development process and outputs. The human-driven approach used the expertise of mid-senior level developers following traditional development practices. The AI-driven approach relied on EngineerGPT, a tool with autonomous software generation capabilities.

Human-Driven Development

The human development team consisted of 2 software engineers with 5 and 7 years of experience. Their expertise included the development of large-scale software applications using the latest web technologies. They were tasked with creating solutions based on the specifications. The developers worked independently of the AI-driven approach. Having used their expertise, they approached the project specifications in a modular and organized manner which resulted in producing complete, cohesive solutions without the need for iterative prompt adjustments. This approach generated the code that was clear, verbose, and highly maintainable along with a strong emphasis on function separation, comments, and additional checks to ensure code readability and future adaptability.

AI-Driven Development with EngineerGPT

The approach used in the AI-driven development that was executed through EngineerGPT faced unique challenges requiring a modular, iterative methodology to optimize the quality of outputs. The initial version of EngineerGPT was based on the GPT-3.5 Turbo model which was later upgraded to GPT-4o. It improved its ability to handle larger functions and more complex requirements.

To address limitations such as output length restrictions and incomplete responses, the development process was restructured into a **modular prompt-testing cycle as in Figure 2**:

1. **Backend for Authentication**: EngineerGPT was first prompted to generate the backend authentication and authorization functions. The outputs were tested against the requirements to validate the output. If incomplete, prompt adjustments were made to ensure they met the necessary criteria. This iterative process continued until the output was functional and aligned with the project specifications.
2. **CGPA & Transcript System**: A similar approach was taken for generating the backend functionalities related to CGPA and transcript management. Each output was assessed for quality and completeness and then the prompts were refined iteratively to produce cohesive and more functional modules.
3. **Frontend Development**: For the frontend development, EngineerGPT generated modules designed to seamlessly integrate with the backend components produced in previous stages. The modular and iterative approach ensured that each component was aligned with the specifications and cohesive within the overall application structure.

This approach, iterative and modular, helped address all the limitations that are inherent to the EngineerGPT while improving consistency and coherence in the output across the application. This development structure allows the AI-driven approach to develop a realistic application structure and therefore enables better comparison between the human-developed solutions.

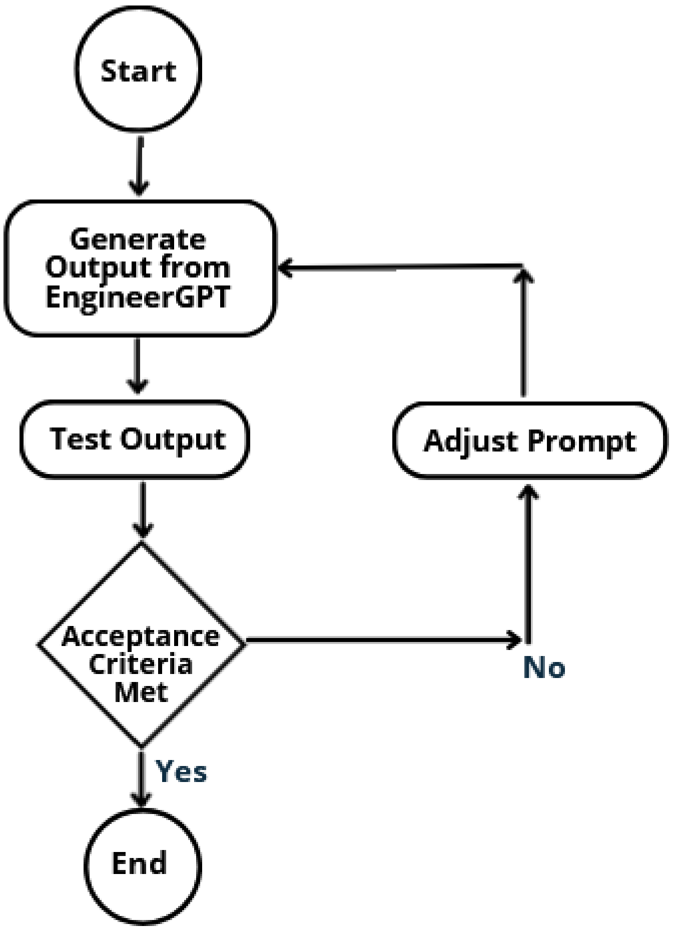


Figure 2: Iterative prompt-testing cycle for refining EngineerGPT outputs

3.2.6. Phase 5: Evaluation Process

In the final phase, a structured evaluation process was established to objectively assess and compare the outputs of human-driven and AI-driven development approaches. The focus of this evaluation is on three primary metrics: feature completeness, code quality, and test case coverage. Using these metrics allowed for a quantitative analysis of the strengths and limitations of each approach along with providing insights into the effectiveness of EngineerGPT compared to human developers.

A. Feature Completeness

The output's implemented features were mapped against the project specifications to assess how well each approach met functional requirements. This focused of this comparison was mainly on the inclusion, functionality, and accuracy of specified features. By using the original specification as a benchmark, feature completeness was evaluated on a quantitative scale, highlighting any gaps or deviations in implementation.

B. Code Quality Assessment

Code quality was measured by static code analysis using SonarCloud. This approach provides objective metrics, standardized, for AI-generated code and human-coded solutions. Prominent indicators of code quality are:

- **Cyclomatic Complexity**: This metric evaluated the complexity of the code, helping to identify potential maintainability challenges.
- **Lines of Code (LOC)**: LOC gave a numerical measurement of the size of the codebase, thus showing how simple or verbose the code was.
- **Code Duplication**: This metric showed duplicate code segments, which represents how much refactoring and optimization was done on each solution.

These metrics allowed for a consistent evaluation of code maintainability, readability, and complexity across both approaches, ensuring a fair comparison.

C. Test Case Coverage

The test cases generated by both AI and human developers were analyzed for quality and comprehensiveness. The focus was on alignment with functional requirements to allow a direct comparison of the effectiveness of each approach in testing.

3.3. Conclusion

This structured evaluation process allowed for an in-depth comparison between human-driven and AI-driven development approaches, employing the same metrics in order to objectify and compare feature completeness, code quality, and test case coverage. The results from this phase proved valuable in examining the practical use and present limitations of AI-driven software development in respect to traditional human development practices.

Chapter 4

RESULTS AND DISCUSSION

4.1. Introduction

This chapter provides a detailed analysis by comparing the outputs of human-driven and AI-driven software development methods. The focus is on key metrics that reflect software quality, feature completeness, and test case coverage. The primary aim is to evaluate the strengths and limitations of AI-driven tools particularly EngineerGPT using the GPT-4o model for its comparison to experience human developers.

As discussed in section 3.2.6, we have selected the metrics of Feature Completeness, Code Quality, and Test Case Coverage. The objective is to provide a comparison of the quality of code produced by both the approaches. However, before moving into these metrics, the chapter begins by addressing the choice of AI tools and models. Specifically, it discusses the exclusion of an AI tool (ChatDev) from the analysis and the selection of the GPT-4o model over GPT-3.5 Turbo. This context is critical as it explains the rationale behind focusing solely on EngineerGPT's GPT-4o outputs for a fair and accurate comparison.

The Exclusion of ChatDev section provides the reasons for which the software produced became unsuitable for evaluation, from missing files to the project structures being incomplete and functionality having considerable gaps. The Model Comparison section outlines improvements when using model GPT-4o instead of GPT-3.5 Turbo with EngineerGPT: improvement in functionalities as well as cohesion in the code being highlighted.

The subsequent sections systematically present the findings:

- **Feature Completeness** evaluates whether all specified features were implemented as per the requirements, comparing the coverage between human and AI-driven approaches.
- **Code Quality** examines structural metrics like cyclomatic complexity, code duplication, and lines of code to assess readability, maintainability, and efficiency.
- **Test Case Coverage** assesses the thoroughness of testing, analyzing how well each approach validated core functionalities and handled potential errors.

Overall, this chapter provides a comprehensive view of how AI-driven software development, using the latest advancements like GPT-4o, measures up to traditional human development. The results contribute to a deeper understanding of AI's role in software engineering and the direction of future research and practical applications.

4.2. Scenarios Implemented in the Study

4.2.1. Scenario 1: CGPA & Transcript Application

Brief Description:
The CGPA & Transcript Application is a web application through which the students will be able to calculate their GPA and to request transcripts. Once students log in, they can find out and calculate their GPA and CGPA and request official transcripts. The admin interface helps view and update the status for submitted transcript applications to make the process smooth and efficient.

Key Features:

- **Student Login**: Secure authentication for students to access GPA and transcript functionalities.
- **GPA Calculation**: Students can input course details, calculate GPA and CGPA, and view the results based on a predefined grading policy.
- **Excel Import**: Students can import course data from an Excel file to simplify GPA calculations.

- **Transcript Application**: Students can submit applications for transcripts, which are stored for administrative processing.
- **Admin Interface**: Admins can log in to view, update, or delete transcript applications and manage the application workflow efficiently.

For both experimental setups, the input is a set of system specifications generated through ChatGPT and fine tuned to make it comprehensive and understandable for both human developers and the EngineerGPT (refer to Appendix A for further details).

Nature of Application: Web-based

Technologies Used:

- **By AI (EngineerGPT)**: Node.js for the backend, and HTML, CSS, and JavaScript for the frontend.
- **By Human Developers**: Node.js for the backend and Angular for the frontend.

Team Size: 2 Developers

Complete specifications of the CGPA & Transcript Application are presented in Appendix A.1.

4.2.2. Scenario 2: To-Do List Application

Brief Description:
The To-Do List Application is a Web-based Task Management System, which allows different users to create, manage, and organize tasks efficiently by implementing the user authentication system so that only the respective users can log in to individualized to-do lists; they can add, edit, complete, and delete tasks and update the Profile Information appropriately.

Key Features:

- **User Authentication**: Secure sign-up, login, and logout functionalities.
- **Add Task**: Users can create tasks, including details like title, due date, and priority level.
- **View Tasks**: Users can view tasks in an organized format.

- **Edit Task**: Users can modify existing tasks and see immediate updates.
- **Mark Task as Completed**: Tasks can be marked as complete, with visual indicators distinguishing completed tasks.
- **Delete Task**: Users can delete tasks, with a confirmation prompt to prevent accidental deletions.
- **Update Profile**: Users can update their personal details, including name, email, and password.

For both experimental setups, the input is a set of system specification generated through ChatGPT and fine-tuned to make it comprehensive and understandable for both human developers and the EngineerGPT (refer to Appendix A.2 for further details).

Nature of Application: Web-based

Technologies Used:

- **By AI (EngineerGPT)**: Node.js for the backend, and HTML, CSS, and JavaScript for the frontend.
- **By Human Developers**: Node.js for the backend and Angular for the frontend.

Team Size: 2 Developers

Complete specifications of the To-Do List Application are presented in Appendix A.2.

All Prompts given to EngineerGPT are presented in Appendix B.

4.3. Selection of Generative AI based Tools

Initially, multiple AI-driven development tools, including ChatDev and EngineerGPT, were considered for their ability to autonomously generate complete software applications. As part of the current study, it was also required to identify the strengths and weaknesses of available generative AI based software development tools. For this purpose, some experimentation was carried out to make comparison of these tools and select an appropriate tool for further study.

For this purpose, a unified input was provided to ChatDev and EngineerGPT. However, after analyzing the output of these tools we decided to proceed with EngineerGPT. The primary reason for the exclusion of ChatDev was its limited capability to understand and effectively implement the provided software requirements.

Key Limitations of ChatDev:

- **Incomplete Project Structure**: ChatDev generated a project with missing files that were needed for proper execution. For example, the package.json file was missing which is crucial for managing dependencies and configuring the application. This incomplete file structure made it difficult to set up and run the software resulting in significantly impacting its usability.
- **Requirement Understanding**: ChatDev also struggled to accurately interpret and implement specific requirements, such as the use of a MySQL database. Explicitly mentioned MySQL as the required database system in the requirements, ChatDev produced a simplified file-based storage solution instead. This deviation from the requirements depicted the generated software inadequate for meeting the specified functional needs.

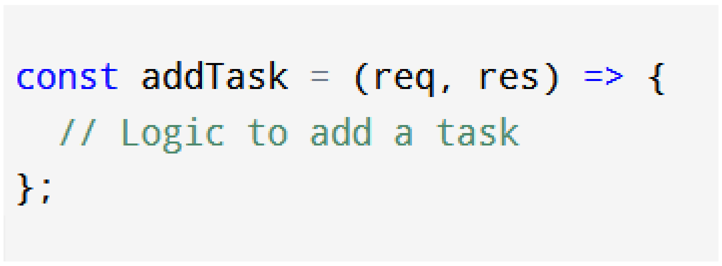
Rationale for Selecting EngineerGPT: EngineerGPT was preferred as the tool for this study as it demonstrated a stronger ability to produce cohesive and functional software. Although there were some of its own limitations, but EngineerGPT's output was more reliable and complete compared to that of ChatDev.

4.4. Comparison of GPT-3.5 Turbo & GPT-4o in EngineerGPT

As the study progressed, there were upgrades available in GPT version as well. GPT was upgraded from version 3.5 to 4o. The differences in terms of the quality of output, accuracy and completeness between GPT-3.5 Turbo and GPT-4o, highlighted the significance of making the comparison between both. Thus, we also included this comparison in our study, particularly in terms of their ability to generate complete and functional backend code for the **To-Do List Application** using EngineerGPT. The comparison highlights the impact of output size limitations and advancements in handling complex requirements.

4.4.1. Code Generation Completeness

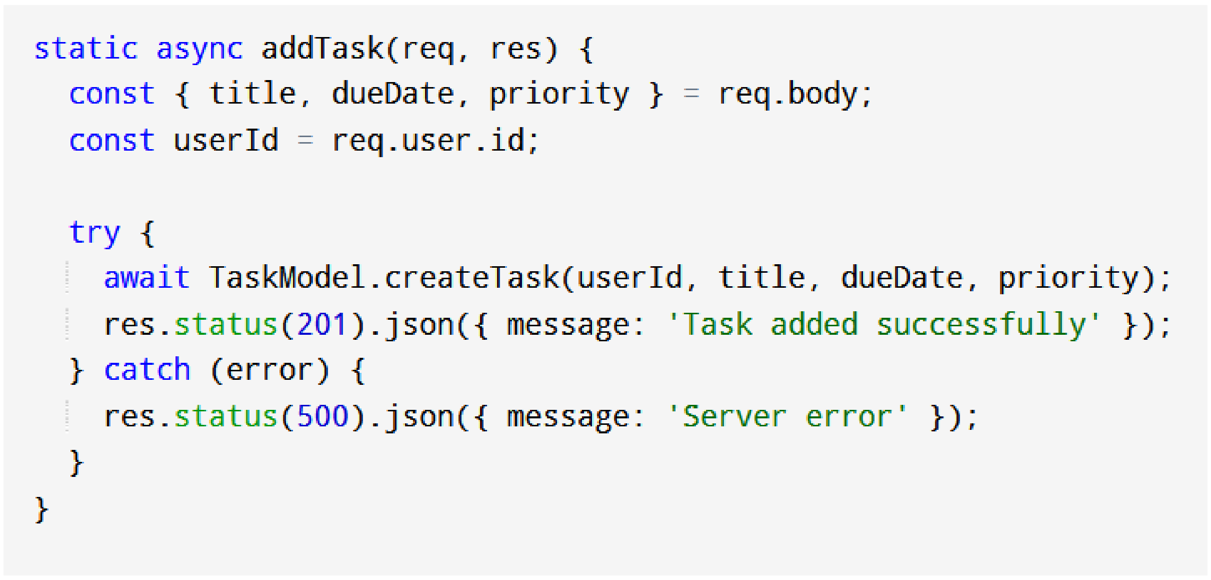
GPT-3.5 Turbo: This model produced code that was structurally sound and well-formed, but it often fell short of delivering complete functionality. The primary limitation was its restricted output size, which resulted in truncated code segments. As a consequence, the model generated function headers and structural outlines but left important logic unimplemented, necessitating human intervention to complete the missing parts. Despite this, the quality of the generated code was not compromised same as GPT-4o, with well-organized syntax and logical structures. *(See Figure 3 for an example of the incomplete code generated by GPT-3.5 Turbo.)*



```
const addTask = (req, res) => {
  // Logic to add a task
};
```

Figure 3: An Example of Incomplete Code Generated by GPT-3.5 Turbo

GPT-4o: The GPT-4o model addressed the output size limitation that enabled the generation of more extensive and cohesive code. It successfully implemented full functions that include the detailed inner logic which reduced the need for additional manual effort. Its ability to generate complete code made the development process smoother and more efficient. *(See Figure 4 for an example of the complete code generated by GPT-4o.)*



```
static async addTask(req, res) {
  const { title, dueDate, priority } = req.body;
  const userId = req.user.id;

  try {
    await TaskModel.createTask(userId, title, dueDate, priority);
    res.status(201).json({ message: 'Task added successfully' });
  } catch (error) {
    res.status(500).json({ message: 'Server error' });
  }
}
```

Figure 4: An Example of Complete Code Generated by GPT-4o

4.4.2. Handling of Complex Requirements:

GPT-3.5 Turbo: With its limitations in the output size, the developers were required to break down complex and larger prompts into smaller and more manageable tasks. The quality of the generated code was of good quality, the need for iterative prompting and combining multiple outputs made the overall development process less efficient. Human developers were required to integrate the outputs together and refine the code to ensure functionality and cohesiveness.

GPT-4o: GPT-4o stands out in handling complex and interdependent requirements. The model has the ability to produce larger, integrated code segments that required fewer iterations and minimal manual effort. With this, the improvement in the workflow was seen as fewer prompts were needed, and the generated code was more cohesive from the start.

4.4.3. Efficiency and Human Intervention:

GPT-3.5 Turbo: Although the code quality was good, but the human efforts were required to for the integration and completion. The output required to fill in missing logic, to merge the different code segments, and make all the parts work together as intended. As such, the overall process is quite inefficient, not due to code errors but just the fragmented nature of the output.

GPT-4o: The need for human intervention was significantly reduced with the usage of model GPT-4o. The model generated more comprehensive and complete code that enabled the developers to proceed with minimal modifications. This efficiency improvement highlighted the benefits of using a model with enhanced output capabilities.

Observations: The transition from GPT-3.5 Turbo to GPT-4o resulted in a more efficient development process. The primary reason is GPT-4o’s ability to generate complete and cohesive code segments. While the quality of code produced by both models was comparable but the GPT-4o's expanded output size allowed smoother implementation with complex features and less manual effort. These advancements show how newer AI models are becoming more practical for real-world software engineering tasks and helping to reduce the workload on human developers.

4.5. Feature Completeness

Feature completeness assesses the extent to which AI-driven (EngineerGPT) and human-driven approach implemented the required features provided in the standardized specifications. This metric evaluates how thoroughly each solution met the project requirements for both the CGPA & Transcript Application and the To-Do List Application based on the exact implementation results and observed limitations.

4.5.1. CGPA & Transcript Application

The CGPA & Transcript Application’s functionality was evaluated based on essential features, such as login, GPA and CGPA calculations, application submission and tracking, and specific admin functionalities. Each approach’s implementation was assessed for accuracy, completeness, and alignment with the requirements.

Table 2: Feature Completeness Comparison for CGPA & Transcript Application

Feature	**Human-Driven Completion**	**AI-Driven Completion**
Student Login	Fully Implemented	Partially Implemented (no role restriction)

Feature	**Human-Driven Completion**	**AI-Driven Completion**
Manual Course Input	Fully Implemented	Fully Implemented
Excel Import	Fully Implemented	Missing
View GPA	Fully Implemented	Partially Implemented (incorrect calculation)
View CGPA	Fully Implemented	Partially Implemented (incorrect calculation)
Submit Transcript Application	Fully Implemented	Fully Implemented
List of Submitted Applications	Fully Implemented	Partially Implemented (missing view for students)
View Applications (Admin)	Fully Implemented	Fully Implemented
Admin Login	Fully Implemented	Partially Implemented (no role restriction)
Update Application Status	Fully Implemented	Fully Implemented
Delete Application	Fully Implemented	Fully Implemented

A. Human-Driven Development:

The human developers achieved full feature implementation, covering all user roles, functionality, and access restrictions. Specific functionalities included:

- **Login Functionality**: Role-based access was correctly implemented, with separate student and admin logins enforcing role-specific access to the portal. Unauthorized routes were restricted, ensuring secure access control.
- **Navigation and UI**: The user interface was intuitive, with separate navigation bars before and after login. A logout button was present for both the student and admin portals.
- **GPA & CGPA Calculations**: GPA and CGPA were calculated accurately, covering a variety of input scenarios, including course-specific and aggregate GPA calculations.
- **Transcript Management**: Students had a dedicated page for submitting and viewing their transcript applications, while admins could view, update, and delete applications.

- **Excel Import**: The Excel import functionality was fully implemented, allowing students to upload course data for streamlined data entry and GPA calculations.

B. AI-Driven Development (EngineerGPT):

EngineerGPT implemented primary features but showed several missing or incomplete elements:

- **Login Functionality**: Role restrictions were absent, allowing both students and admins to access each other's portals, which compromised security and access control.
- **Navigation and UI**: The same navigation bar was displayed before and after login, exposing unauthorized routes to all users. Additionally, a logout button was missing in both portals.
- **GPA & CGPA Calculations**: The AI-driven application had issues with accurate GPA and CGPA calculations, failing to produce the correct results in many cases.
- **Transcript Management**: Students lacked a dedicated page to view their submitted transcript applications, limiting their ability to track application status.
- **Excel Import**: This feature was entirely missing, restricting students to manual data entry without the option to upload course data via file.

Observations: The human-driven approach ensured complete feature alignment with the requirements, incorporating role-specific access, accurate GPA/CGPA calculations, a structured user interface, and Excel import functionality. EngineerGPT, while effective in implementing basic functionality, lacked advanced access control, accurate calculations, and the Excel import feature, limiting its ability to fully meet project specifications.

4.5.2. To-Do Application

For the To-Do Application, the evaluation criteria focused on user authentication, task management, profile updates, and error handling. Each feature's implementation status was checked against the specifications for completeness and functionality.

Table 3: Feature Completeness Comparison for To-Do Application

Feature	**Human-Driven Completion**	**AI-Driven Completion**
Sign-up	Fully Implemented	Partially Implemented (missing password validation)
Login	Fully Implemented	Fully Implemented
Logout	Fully Implemented	Fully Implemented
Add Task	Fully Implemented	Fully Implemented
View Tasks	Fully Implemented	Fully Implemented
Edit Task	Fully Implemented	Fully Implemented
Mark Task as Completed	Fully Implemented	Fully Implemented
Delete Task	Fully Implemented	Fully Implemented
Update Profile	Fully Implemented	Partially Implemented (inaccessible route in UI)
Error Handling (password)	Robust, covers complexity	Limited, no error on low-complexity password

A. Human-Driven Development:

The human developers achieved near-complete feature implementation, ensuring robust functionality across the board. Highlights included:

- **User Authentication:** Fully implemented, with signup, login, and logout capabilities. The system enforced password complexity and validated user credentials thoroughly, including duplicate email detection.
- **Task Management:** Task creation, editing, marking as completed, and deletion were all covered with extensive test cases, ensuring a smooth user experience.
- **Profile Updates:** A dedicated route was developed that allowed users to update their profiles. It also included email uniqueness checks to prevent duplicate entries.
- **Error Handling:** Strong error handling was put in place that included the validation for password complexity and user-friendly error messages for invalid data.

B. AI-Driven Development (EngineerGPT):

EngineerGPT covered core functionalities but showed limitations in user experience and error handling.

a. **User Authentication:** Basic signup, login, and logout features were implemented, but there were issues with password complexity validation, as low-complexity passwords did not trigger error messages, resulting in undefined responses.
b. **Task Management:** Core task functions, including add, edit, mark as completed, and delete, were implemented, covering essential task management requirements.
c. **Profile Updates:** The profile update functionality was available by navigating to /profile, but the route was not integrated into the user interface, requiring users to enter the URL manually.
d. **Error Handling:** Error handling was limited, particularly for password validation during signup, where low-complexity passwords were accepted without an error message.

Observations: Human developers provided comprehensive functionality and error handling, aligning fully with project requirements and ensuring robust user experience. For EngineerGPT, it had achieved core task management features but with issues about access to routes for the profile, error handling, and password validation, which were impacting not only the user experience but also security.

4.5.3. Overall Analysis for Feature Completeness:

To address the RQ1, the extent to which AI-generated software and human-developed software align with the provided specifications in terms of feature implementation. The analysis of both the CGPA & Transcript Application and the To-Do List Application shows distinct differences in how effectively each approach met project requirements.

CGPA & Transcript Application: The human-driven development process provided a solution that met all specified features comprehensively. Human developers implemented the required role-based access control, provided accurate GPA/CGPA calculations, and a structured user interface, ensuring that each requirement was fulfilled. Additionally, the requirement of the Excel import functionality is also fulfilled to enhance data handling capabilities and aligning closely with user needs. For EngineerGPT, it succeeded in establishing basic functionality, it showed limitations in several advanced areas. Specifically, its output lacked the required role-based access control that resulted in restricted security management. Moreover, issues with accurate GPA/CGPA calculations and the absence of the

Excel import feature highlighted gaps in its ability to fully align with the provided specifications. All this indicates that complex, multi-step functionalities remain a challenge for the AI-driven approach.

To-Do List Application: Human developers produced a robust, user-focused solution for the To-Do List Application. All the requirements were met, including comprehensive task management, error handling, profile management, and password validation, which contributed to a secure and reliable user experience. In comparison, EngineerGPT implemented only the core task management features effectively. It showed deficiencies in key areas. Specifically, it lacked error handling along with the limited password validation. Another missing part found was an inaccessible profile route in the user interface. All these shortcomings have impacted the application's security, usability, and overall completeness.

Summary: The comparative analysis proves that human-driven software development performs better in developing the complete feature alignment with the project specifications. Especially in complex requirements involving access control, error handling, and even importing data. While EngineerGPT can effectively generate core functionalities, it struggles with advanced requirements and complicated feature integrations. This therefore indicates that even though AI-driven tools are powerful, still human expertise is crucial to meet fully and specification-compliant software solution, especially where more complex feature implementation and security measures are required for application.

These insights directly answer RQ1, showing that AI tools are valuable for generating basic features but still lack the capabilities to equal the full spectrum of capabilities and precision offered by human developers in aligning software outputs with detailed project requirements.

4.6. Code Quality

The Code quality analysis explores the structural and maintainability aspects of the output for AI-driven and human-driven development. Using SonarCloud, to quantitatively compare the readability, modularity, and optimizing approach through cyclomatic complexity, lines of code, and duplication in the outputs. These metrics reflect long-term maintainability and adaptability of the generated code.

4.6.1. Cyclomatic Complexity

Cyclomatic complexity is a count of the number of independent paths through a program's source code. Lower cyclomatic complexity generally means more maintainable, simpler code; greater values may indicate complex logic that requires additional maintenance efforts.

Table 4: Cyclomatic Complexity of CGPA & Transcript Application

Metric	**Human-Developed System**	**AI-Developed System**
Cyclomatic Complexity	480	108

The human developers produced a codebase with a cyclomatic complexity of 480, indicating a detailed control structure and well-defined functionality. This higher complexity is a result of their modular design approach, where specialized functions and modular pathways were created to ensure thorough coverage of application logic while also promoting maintainability.

EngineerGPT produced a codebase with cyclomatic complexity at 108. That means this one is less complex in design; the AI was aiming for direct functional implementation without more modular breakdowns. While it is less complex, it may also imply a less flexible code structure that has fewer modularized elements to adequately support handling mechanisms of complex logics.

Table 5: Cyclomatic Complexity for To-Do Application

Metric	**Human-Developed System**	**AI-Developed System**
Cyclomatic Complexity	292	63

For human-driven development, the cyclomatic complexity was 292 that reflects the human developers modular structuring and effort to cover complex control flows within the application.

In this case the EngineerGPT achieved a cyclomatic complexity of 63 for the To-Do Application. This simpler complexity level highlights the AI’s concise code generation but suggests a potential trade-off in modular control, with fewer pathways to manage complex behaviors.

Observations: Human developers consistently produced code with higher complexity due to intentional modularization, which enhances flexibility and supports future expansion of the code structure. On the other hand, EngineerGPT’s outputs, while effective, followed a more linear approach with fewer control paths. This results in a codebase that is potentially less adaptable to future additions or changes in functionality.

4.6.2. Lines of Code (LOC)

The Lines of Code (LOC) metric measures the verbosity of the codebase. A high LOC can indicate redundancy or unnecessary complexity, whereas a lower LOC generally suggests a more concise and streamlined code structure, as long as readability and functionality are not sacrificed. Tables 6 and 7 display the LOC produced for both applications by human developers and EngineerGPT, offering a comparison of their respective codebases.

Table 6: Lines of Code (LOC) for CGPA & Transcript Application

Metric	**Human-Developed System**	**AI-Developed System**
Lines of Code (LOC)	13,353	883

As shown in Table 6, the human-developed CGPA application has a total of 13,353 LOC, reflecting a detailed and thoughtful approach. This includes extensive use of frameworks, libraries, and themes, which contribute to the initial increase in LOC. The code is also organized with modular function separation and thorough documentation, including comments and distinct functions, all aimed at improving readability and maintainability. These added elements result in a comprehensive, yet more verbose, codebase.

EngineerGPT produced a notably more compact codebase, with an LOC of just 883. This concise output reflects the AI's focus on directly implementing functionality, without the inclusion of additional structural elements or documentation layers, such as modular comments or helper libraries. As a result, the code structure is streamlined and more efficient.

Table 7: Lines of Code (LOC) for To-Do Application

Metric	**Human-Developed System**	**AI-Developed System**
Lines of Code (LOC)	13,088	624

For the To-Do application, Table 7 presents a comparative analysis of the Lines of Code (LOC) produced by both approaches. The human-developed To-Do application, with a total of 13,088 LOC, reflects a similarly comprehensive approach. The inclusion of frameworks, themes, and additional libraries contributed to the increase in LOC. Although the codebase is large, it is designed with extensive documentation and modular functions, which enhances clarity and ensures adaptability for future modifications.

EngineerGPT generated a compact codebase with an LOC of 624. By focusing solely on the essential functional elements, the AI's approach resulted in fewer lines, with minimal added documentation or modular structuring.

Observations: The higher LOC in human-developed systems reflects the inclusion of frameworks, libraries, and thematic elements, which create a more extensive starting point. This approach, combined with detailed function definitions and documentation, contributes to higher LOC. Conversely, EngineerGPT's compact codebases suggest a minimalistic style, prioritizing functional implementation with little extraneous documentation, which can enhance initial development speed but may limit code readability and adaptability for future maintenance.

4.6.3. Code Duplication

Code duplication indicates redundancy within the codebase, which can affect maintainability and efficiency. Higher duplication may suggest a lack of optimization, while minimal duplication generally reflects a reusable and modular code structure.

Table 8: Code Duplication (Lines) for CGPA & Transcript Application

Metric	**Human-Developed System**	**AI-Developed System**
Code Duplication (lines)	563	32

As demonstrated in Table 8, in human-driven development, code duplication was 563 lines, often resulting from repeated code blocks for additional context and clear task separation. While this duplication enhances readability, it also contributes to a larger codebase.

However, EngineerGPT achieved low duplication with only 32 lines repeated, indicating a more concise coding approach. The AI focused on producing unique, non-repetitive code structures, which reduced redundancy.

Table 9: Code Duplication (Lines) for To-Do Application

Metric	**Human-Developed System**	**AI-Developed System**
Code Duplication (lines)	145	0

For the To-Do list, the same pattern can be observed for code duplication as demonstrated earlier. For human-driven development, the duplication count was 145 lines, where additional context was added through repeated code for clarity and separation of functions. While, EngineerGPT achieved 0 duplication, highlighting an efficient and optimized approach to code generation.

Observations: Human developers had higher duplication to ensure clear function separation, enhancing readability. Conversely, EngineerGPT’s outputs exhibited minimal duplication, producing more efficient, reusable code, which may support maintainability in projects with straightforward requirements. However, as part of our study, there was no observation or requirements put on the human developers to constantly refactor their code. It is possible, that if human developers had adopted and constant approach of refactoring, code duplication could have been reduced.

4.6.4. Overall Analysis for Code Quality

Cyclomatic Complexity

In both the CGPA & Transcript Application and the To-Do Application, human developers consistently produced higher Cyclomatic Complexity values (480 and 292, respectively) compared to EngineerGPT (108 and 63). This increased complexity reflects human developers' modular approach, with more pathways and discrete functions created to support nuanced control and flexible functionality. While this complexity may contribute to easier updates and feature additions, it also results in a codebase that requires more effort to maintain. Conversely, EngineerGPT's lower complexity scores indicate a streamlined, linear approach with limited modularization, which may simplify initial development but potentially restrict flexibility for future expansions or adjustments.

Lines of Code (LOC)

The LOC for human-developed applications was significantly higher (13,353 and 13,088) compared to the AI-developed codebases (883 and 624). This discrepancy is attributed to the human developers' inclusion of frameworks, libraries, and detailed modular functions that inherently expand the LOC. This verbosity, while increasing the codebase size, provides a clear structure, thorough documentation, and enhances readability, supporting long-term maintainability. Conversely, EngineerGPT came up with a compact codebase that contains less documentation and modularization, which encourages quick launching but could be a problem when scalability and flexibility are necessary in complex designs, where detailed separation of function would be more significant.

Code Duplication

Human-driven code showed higher duplication levels (563 and 145 lines) compared to EngineerGPT's low to zero duplication (32 and 0 lines). Human developers employed repeated code blocks to reinforce function separation and improve readability. This redundancy, although making the codebase larger, supports clear contextualization of tasks and enhances understanding for future developers. As discussed before, as part of our study, there was no observation or requirements put on the human developers to constantly refactor their code. It is possible that if human developers had adopted and constant approach of refactoring, code duplication could have been reduced. In contract, EngineerGPT produced low-duplicate, well-optimized code with a bias towards unique implementations for each task, which would minimize the amount of code a smaller project had to maintain but was too low in duplication for intensive systems that could make the overall code base not clear enough where more context was required.

The code quality assessment emphasizes the different approaches taken by human-driven and AI-driven development. Human developers focus on readability, modularity, and maintainability, creating a codebase that is detailed and well-organized, ideal for complex and adaptable applications. Though resource-intensive, this approach ensures long-term flexibility. On the other hand, EngineerGPT generates code that is compact and efficient, reflecting a straightforward implementation style. While this minimizes complexity and reduces duplication, it may not offer the same level of adaptability for future changes and maintenance, especially in larger-scale projects.

These findings suggest that, while EngineerGPT can produce functional and efficient code quickly, human-driven development remains advantageous for projects requiring robust modularity, detailed documentation, and ease of future enhancement. Thus, for RQ2, it is clear that the AI tools like EngineerGPT currently provide an efficient but less flexible solution, while human developers develop code bases with higher scalability and maintainability potential, and thus are better suited for complex, moving software engineering projects.

4.7. Test Case Coverage

This section compares the test cases from both human-driven and AI-driven approaches (using EngineerGPT), aligning them directly with the specified features of each application. The tables below illustrate the test case coverage for each feature, offering a straightforward comparison of how complete the coverage is for each approach.

Test Cases developed by EngineerGPT and Human Developers are presented in Appendix C.

4.7.1. CGPA & Transcript Application Test Case Coverage

The table below maps test cases developed by human developers and EngineerGPT for the CGPA & Transcript Application, focusing on required features.

Table 10: Test Case Coverage for CGPA & Transcript Application

Feature/Requirement	**Human-Driven Coverage**	**AI-Driven Coverage**
Student Login	Yes	Yes
Manual Course Input	Yes	Yes
Excel Import	Yes	Yes
View GPA	Yes	Yes
View CGPA	Yes	Yes
Submit Transcript Application	Yes	Yes
List of Submitted Applications	Yes	No
Admin Login	Yes	Yes
View Applications (Admin)	Yes	Yes
Update Application Status	Yes	Yes
Delete Application	Yes	Yes

Observations: The human-driven approach provided complete coverage for all required features in the CGPA & Transcript Application, addressing both student and admin functionalities comprehensively. EngineerGPT also covered most core features but lacked test cases for the List of Submitted Applications feature, which limited students' ability to view their submitted applications. *(See Test Case 6 below, for an example of the test case missed by AI tool, but generated by Human Developer)*

Test Case 6: View Submitted Transcript Applications (Student)

- **Test Case ID:** TC_006
- **Description:** Verify that a student can view all its submitted transcript applications.
- **Preconditions:** Student is logged in, and there are submitted applications.
- **Test Steps:**

1. Navigate to the transcript request page.
2. View the list of submitted applications.

- **Expected Result:** The system should display all submitted transcript applications with relevant details.
- **Postcondition:** Student can view and access all submitted applications.

4.7.2. To-Do Application Test Case Coverage

The table below compares test case coverage for the required features in the To-Do Application.

Table 11: Test Case Coverage for To-Do Application

Feature/Requirement	Human-Driven Coverage	AI-Driven Coverage
Sign-up	Yes	Yes
Login	Yes	Yes
Logout	Yes	Yes
Add Task	Yes	Yes
View Tasks	Yes	Yes
Edit Task	Yes	Yes
Mark Task as Completed	Yes	Yes
Delete Task	Yes	Yes
Update Profile	Yes	Yes

Observations: For the To-Do Application, both human developers and EngineerGPT covered all specified features comprehensively, including user authentication, task management, and profile updates.

4.7.3. Overall Analysis for Test Case Coverage

This analysis shows that EngineerGPT is capable of generating effective test cases for core functionalities but sometimes its coverage may lack the thoroughness needed to address all feature requirements. This is evident particularly in more complex applications with multiple user roles and access levels. In Contract, the Human-driven test cases did consistently ensure full coverage and supporting thorough validation of both primary and secondary features. For RQ3, the findings indicate that while AI-driven tools like EngineerGPT can offer strong test coverage for key features, human developers still have the upper hand when it comes to creating comprehensive and detailed test cases that are crucial for validating every aspect of complex applications.

Chapter 5

CONCLUSION AND FUTURE WORK

This chapter concludes the findings and outcomes of the current work by first summarizing the outcomes and later providing future directions for further enhancing this work.

5.1. Key Findings

This research conducted a comparative analysis between AI-driven software development tools (specifically **EngineerGPT**) and human developers (mid-senior level software engineers) across two real-world projects: a **CGPA & Transcript Application** and a **To-Do Application**. The objective was to evaluate whether AI can autonomously generate software systems of comparable quality and completeness to those developed by human engineers.

Key findings from the research include:

1. **Inconsistency in AI Output**:

   While **EngineerGPT** was capable of generating functional software systems in a matter of minutes, it required multiple attempts (4 to 5 on average) to produce an acceptable output for the same set of requirements. This highlights the inconsistency in AI-generated code, a limitation that contrasts with the more consistent performance of human developers.

2. **Limitations in Handling Larger Systems**:

AI tools like **EngineerGPT** sometimes struggled or even crashed when tasked with generating larger systems that require substantial output. Even when the system was developed, it often required many changes, and many key requirements were missed. A recommended approach to mitigate this issue is to break down larger systems into manageable modules such as **authentication services**, **CGPA and transcript backend services**, and **frontend applications**. This modular approach not only reduces the risk of tool failure but also aligns with common software engineering best practices, improving the chances of feature completeness.

3. **Speed vs Completeness**:

One of the most significant advantages of AI-based development was the speed with which software was produced. However, this came at the cost of feature completeness. The AI system often produced incomplete features, whereas human developers implemented all required features as per the specification. This reflects a trade-off where AI excels in rapid generation but struggles with feature completeness and complex functionality.

4. **Code Quality**

The AI-generated code was generally more concise and exhibited lower cyclomatic complexity and code duplication compared to human-written code. While this suggests that AI can produce efficient code, the simplicity may also indicate oversimplification in logic, leading to missing or incomplete features. Human developers, although generating more complex and lengthier code, produced more robust and functional software systems.

5. Test Coverage

EngineerGPT demonstrated reasonable capabilities in generating test cases and achieving a relatively high percentage of test coverage, with over 90% in most cases. However, human developers were able to achieve complete test coverage and develop more comprehensive test cases that effectively identified potential defects.

6. **Human Expertise and AI Augmentation**

The human developers involved in this study were mid-senior level engineers. Their expertise enabled them to handle complex logic and feature implementation, problem-solving tasks where AI struggled. This indicates that AI tools, while useful, still require human oversight, particularly in handling more intricate software development tasks and ensuring the completeness of the system.

In summary, **EngineerGPT** and similar AI-driven tools demonstrate considerable potential for automating routine or repetitive tasks and accelerating software development. However, they fall short in achieving feature completeness, consistency, and handling complex requirements, which still require human intervention. Human developers produced more reliable, functional, and feature-complete systems, though at a slower pace.

5.2. Future Work

Although this research provides valuable insights into the capabilities and limitations of AI-driven software development tools, several areas for future work remain:

1. **Comparison of Different Experience Levels in Human Developers**

   Future research could focus on comparing AI-driven development tools with human developers of varying experience levels (junior, mid-level, and senior). This would provide a deeper understanding of where AI tools may provide the most value and whether they can effectively augment or support less experienced developers.

2. **Human-AI Collaboration and Modular Development:**

   As noted in this study, AI tools sometimes struggle or crash when tasked with generating large systems in a single iteration, and even when successful, they often miss key requirements. Future work could focus on breaking down large software systems into smaller, manageable modules, such as authentication services, backend services, and frontend applications. This modular approach would enable AI tools to handle the task more effectively while reducing the likelihood of crashes or incomplete functionality. Further research should explore how AI can work in collaboration with human developers in a modular system to maximize productivity and completeness.

3. **Broader Range of Software Applications**

This study was limited to two software applications. Future research could explore the performance of AI tools across a broader range of software systems, including larger-scale enterprise applications or domain-specific systems (e.g., healthcare, finance, or cybersecurity).

4. **Improving AI Consistency and Feature Completeness**

One of the key limitations of **EngineerGPT** was its inconsistency and inability to fully implement required features on the first attempt. Future research should focus on improving the underlying models to produce more consistent and feature-complete software outputs, potentially through better training, prompt engineering, or iterative refinement.

5. **Security and Reliability of AI-Generated Software**

This study did not focus on the security and reliability of AI-generated code. Future work should assess the security vulnerabilities and scalability of AI-generated systems, particularly when deployed in real-world scenarios.

6. **Ethical Implications of AI in Software Development**

As AI tools become more prevalent in the software development landscape, there are important ethical considerations to address. Future research could explore the implications of AI on the job market for developers, intellectual property concerns, and how responsibility is assigned when AI-generated code fails or causes harm.

7. **Longitudinal Studies on AI's Impact on Development Practices**:

Conducting long-term studies on the integration of AI tools in development workflows could provide valuable insights into how AI changes productivity, team dynamics, and software quality over time. This would help in understanding the broader impact of AI on the software engineering profession.

5.3. Conclusion

This research has demonstrated that while AI tools like **EngineerGPT** can accelerate certain aspects of software development and produce efficient code, they are not yet ready to fully replace human developers in complex projects. The most effective future approach may lie in **hybrid development environments**, where AI assists human developers in routine tasks, while human expertise remains essential for handling complex features, problem-solving, and ensuring overall software quality.

As AI technologies continue to evolve, it will be critical for the software engineering field to embrace these tools in ways that enhance productivity and quality while maintaining human oversight for critical and complex development tasks.

References

[1] H. Sofian, N. A. M. Yunus, and R. Ahmad, “Systematic Mapping: Artificial Intelligence Techniques in Software Engineering,” *IEEE Access*, vol. 10, pp. 51021–51040, 2022, doi: 10.1109/ACCESS.2022.3174115.

[2] I. Ozkaya, “The Next Frontier in Software Development: AI-Augmented Software Development Processes,” Jul. 01, 2023, *IEEE Computer Society*. doi: 10.1109/MS.2023.3278056.

[3] X. Hou *et al.*, “Large Language Models for Software Engineering: A Systematic Literature Review,” Aug. 2023, [Online]. Available: http://arxiv.org/abs/2308.10620

[4] D. Russo, “Navigating the Complexity of Generative AI Adoption in Software Engineering,” *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 5, Jun. 2024, doi: 10.1145/3652154.

[5] R. Feldt, F. G. de O. Neto, and R. Torkar, “Ways of Applying Artificial Intelligence in Software Engineering,” Feb. 2018, [Online]. Available: http://arxiv.org/abs/1802.02033

[6] A. Nguyen-Duc *et al.*, “Generative Artificial Intelligence for Software Engineering -- A Research Agenda,” Oct. 2023, [Online]. Available: http://arxiv.org/abs/2310.18648

[7] N. Nascimento, P. Alencar, and D. Cowan, “Comparing Software Developers with ChatGPT: An Empirical Investigation,” May 2023, [Online]. Available: http://arxiv.org/abs/2305.11837

[8] N. Nascimento, P. Alencar, D. Cowan, and D. R. Cheriton, “Artiicial Intelligence versus Software Engineers: An Evidence-Based Assessment Focusing on Non-Functional Requirements Artificial Intelligence versus Software Engineers: An Evidence-Based Assessment Focusing on Non-Functional Requirements,” 2023, doi: 10.21203/rs.3.rs-3126005/v1.

[9] S. Imai, “Is GitHub copilot a substitute for human pair-programming?,” Association for Computing Machinery (ACM), May 2022, pp. 319–321. doi: 10.1145/3510454.3522684.

[10] A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, M. S. Aktar, and T. Mikkonen, “Towards Human-Bot Collaborative Software Architecting with ChatGPT,” in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Jun. 2023, pp. 279–285. doi: 10.1145/3593434.3593468.

[11] J. Sauvola, S. Tarkoma, M. Klemettinen, J. Riekki, and D. Doermann, “Future of software development with generative AI,” *Automated Software Engineering*, vol. 31, no. 1, May 2024, doi: 10.1007/s10515-024-00426-z.

[12] S. L. France, “Navigating software development in the ChatGPT and GitHub Copilot era,” *Bus Horiz*, vol. 67, no. 5, pp. 649–661, Sep. 2024, doi: 10.1016/j.bushor.2024.05.009.

[13] N. A. Ernst and G. Bavota, “AI-Driven Development Is Here: Should You Worry?,” *IEEE Softw*, vol. 39, no. 2, pp. 106–110, 2022, doi: 10.1109/MS.2021.3133805.

[14] A. Fan *et al.*, “Large Language Models for Software Engineering: Survey and Open Problems,” Oct. 2023, [Online]. Available: http://arxiv.org/abs/2310.03533

[15] A. Mastropaolo *et al.*, “On the Robustness of Code Generation Techniques: An Empirical Study on GitHub Copilot,” in *Proceedings - International Conference on Software Engineering*, IEEE Computer Society, 2023, pp. 2149–2160. doi: 10.1109/ICSE48619.2023.00181.

[16] L. Korada, “GitHub Copilot: The Disrupting AI Companion Transforming the Developer Role and Application Lifecycle Management,” *Journal of Artificial Intelligence & Cloud Computing*, vol. 3, no. 3, pp. 1–4, Jun. 2024, doi: 10.47363/JAICC/2024(3)348.

[17] C. Bird *et al.*, “Taking Flight with Copilot,” *Commun ACM*, vol. 66, no. 6, pp. 56–62, May 2023, doi: 10.1145/3589996.

[18] R. Pudari and N. A. Ernst, “From Copilot to Pilot: Towards AI Supported Software Development,” Mar. 2023, [Online]. Available: http://arxiv.org/abs/2303.04142

[19] L. Wang *et al.*, “A Survey on Large Language Model based Autonomous Agents,” Aug. 2023, [Online]. Available: http://arxiv.org/abs/2308.11432

[20] M. Barenkamp, J. Rebstadt, and O. Thomas, “Applications of AI in classical software engineering,” *AI Perspectives*, vol. 2, no. 1, Dec. 2020, doi: 10.1186/s42467-020-00005-4.

[21] F. A. Batarseh, A. Kumar, R. Mohod, and J. Bui, “Chapter 10: The Application of Artificial Intelligence in Software Engineering-A Review Challenging Conventional Wisdom.”

[22] A. Shastri Pothukuchi *et al.*, “IMPACT OF GENERATIVE AI ON THE SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC),” 2023. [Online]. Available: www.ijcrt.org

[23] I. Ozkaya, “Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications,” May 01, 2023, *IEEE Computer Society*. doi: 10.1109/MS.2023.3248401.

[24] A. M. Dakhel *et al.*, “GitHub Copilot AI pair programmer: Asset or Liability?,” Jun. 2022, [Online]. Available: http://arxiv.org/abs/2206.15331

[25] B. Zhang, P. Liang, X. Zhou, A. Ahmad, and M. Waseem, “Practices and Challenges of Using GitHub Copilot: An Empirical Study,” in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, Knowledge Systems Institute Graduate School, 2023, pp. 124–129. doi: 10.18293/SEKE2023-077.

[26] Y. Dong, X. Jiang, Z. Jin, and G. Li, “Self-collaboration Code Generation via ChatGPT,” Apr. 2023, [Online]. Available: http://arxiv.org/abs/2304.07590

[27] M. Latinovic and V. Pammer-Schindler, “Automation and artificial intelligence in software engineering: Experiences, challenges, and opportunities,” in *Proceedings of the Annual Hawaii International Conference on System Sciences*, IEEE Computer Society, 2021, pp. 146–155. doi: 10.24251/hicss.2021.017.

[28] I. Ozkaya, “A Paradigm Shift in Automating Software Engineering Tasks: Bots,” 2022, *IEEE Computer Society*. doi: 10.1109/MS.2022.3167801.

[29] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative Agents for Software Development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186, Bangkok, Thailand. Association for Computational Linguistics.

[30] White, J., Hays, S., Fu, Q., Spencer-Smith, J., Schmidt, D.C. (2024). ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design. In: Nguyen-Duc, A., Abrahamsson, P., Khomh, F. (eds) Generative AI for Effective Software Development. Springer, Cham. https://doi.org/10.1007/978-3-031-55642-5_4

[31] S. Haque, Z. Eberhart, A. Bansal and C. McMillan, "Semantic Similarity Metrics for Evaluating Source Code Summarization," in 2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC), Pittsburgh, PA, USA, 2022, pp. 36-47, doi: 10.1145/nnnnnnn.nnnnnnn.

[32] Tian, H., Lu, W., Li, T. O., Tang, X., Cheung, S. C., Klein, J., & Bissyandé, T. F. (2023). Is ChatGPT the ultimate programming assistant--how far is it?. *arXiv preprint arXiv:2304.11938*.

[33] Zheng, Z., Ning, K., Chen, J., Wang, Y., Chen, W., Guo, L., & Wang, W. (2023). Towards an understanding of large language models in software engineering tasks. *arXiv preprint arXiv:2308.11396*.

[34] Shin, J., Tang, C., Mohati, T., Nayebi, M., Wang, S., & Hemmati, H. (2023). Prompt engineering or fine tuning: An empirical assessment of large language models in automated software engineering tasks. *arXiv preprint arXiv:2310.10508*.

APPENDIX A: REQUIREMENT SPECIFICATIONS

A.1 Requirement Specification for CGPA & Transcript Application

Project Overview

Develop a comprehensive GPA calculator system for students that includes user authentication, course input for GPA calculation, and an admin interface for managing submitted transcript applications. The system should be robust and modular, integrating all functionalities into a single application.

User Stories

For Students

1. **Login:**
    a. As a student, I want to log in using my credentials so that I can access my GPA calculation and transcript application functionalities.
2. **Calculate GPA:**
    a. As a student, I want to input my course details (course name, course code, credit hrs, marks, semester) so that I can calculate my GPA & CGPA. I shall be able to see the generated grade points and grades for each course as per BU grading policy.
    b. As a student, I want to see the calculated GPA after submitting my course details.
    c. As a Student, I shall be able import the required information about courses, credit hrs, marks, semester etc. in an excel file.
3. **Submit Transcript Application:**
    a. As a student, I want to submit an application for my transcript so that I can receive it from the exam department.

For Admins (Exam Department)

1. **Login:**
    a. As an admin, I want to log in using my credentials so that I can manage transcript applications.
2. **View Applications:**
    a. As an admin, I want to view all submitted transcript applications so that I can process them.
3. **Update Application Status:**
    a. As an admin, I want to update the status of a transcript application (Underprocess, Ready to Collect) so that students are informed about the progress.
4. **Delete Application:**
    a. As an admin, I want to delete an application so that I can manage the application records efficiently.

Functional Requirements

Authentication and Authorization

1. The system should support user registration and login using JWT for authentication.
2. Users should be able to log in using their username and password.
3. JWT tokens should be used to authenticate requests to the backend.

GPA Calculation

1. The GPA calculation page should allow students to input multiple courses with their respective grades and credits.
2. The system should calculate the GPA based on the input and display the result.
3. The calculation should convert letter grades to grade points using a predefined scale.

Transcript Application Submission

1. The system should allow students to submit their transcript applications.
2. The application form should include fields for student name, student ID, program, and contact information.
3. Submitted applications should be stored in the database for admin review.

Admin Interface

1. Admins should have a separate login interface.
2. Admins should be able to view all submitted transcript applications.
3. Admins should be able to update the status of an application.
4. Admins should be able to delete an application.

Grading Scale for Bahria University

Bahria University follows a specific grading scale to assign grade points to letter grades. Here is the grading scale:

Letter Grade	Grade Points	Marks
A	4.0	85
A-	3.7	80
B+	3.3	75
B	3.0	71
B-	2.7	68
C+	2.3	64
C	2.0	60
C-	1.7	57
D+	1.3	53
D	1.0	50
F	0.0	0-49

A.2 Requirement Specification for To-Do List Application

1. Project Overview
The objective of this project is to develop a To-Do List application that allows users to create, manage, and organize their tasks efficiently. The system will include essential user authentication features like sign up, login, and logout, ensuring secure access to user-specific tasks. Each user will have their personalized to-do list, with options to add, edit, mark tasks as completed, and delete tasks.

2. Functional Requirements
2.1 User Authentication
2.1.1 Signup
- **Description:** Users must be able to register for an account using their email and password.
- **Requirements:**
  - A registration form must be provided for users to input their details (e.g., name, email, password).
  - Email must be unique for each user.
  - Passwords must meet security criteria (minimum length, use of alphanumeric characters).
  - Provide validation for all form fields (e.g., valid email format, non-empty fields).

2.1.2 Login
- **Description:** Users must be able to log in with valid credentials.
- **Requirements:**
  - A login form must be provided for users to enter their email and password.
  - Passwords must be hashed before being stored in the database.
  - Upon successful login, users should be redirected to their personal dashboard.
  - Invalid credentials should result in appropriate error messages.

2.1.3 Logout
- **Description:** Users should be able to log out of the application securely.
- **Requirements:**
  - Users should have an option to log out.
  - Session or JWT tokens should be invalidated on logout.

2.2 Task Management
2.2.1 Add Task
- **Description:** Users must be able to add tasks to their to-do list.
- **Requirements:**
  - Users should see a form or input field where they can enter task details (e.g., task title, due date, priority).
  - Tasks must be associated with the logged-in user.
  - The system should allow optional fields for due date and priority level (e.g., High, Medium, Low).

2.2.2 View Tasks
- **Description:** Users must be able to view all of their tasks in a structured manner.
- **Requirements:**
  - Tasks should be displayed in a list or table format.

    - Tasks can be filtered by status (e.g., pending, completed) or sorted by due date, priority, etc.
    - Completed tasks should be visually distinguishable from pending tasks.

2.2.3 Edit Task

- **Description:** Users must be able to edit existing tasks.
- **Requirements:**
    - Users should have the option to modify the task title, due date, or priority level.
    - Changes should be saved and reflected immediately.

2.2.4 Mark Task as Completed

- **Description:** Users must be able to mark a task as completed.
- **Requirements:**
    - Users should be able to click a button or checkbox to mark a task as completed.
    - The task should visually update to indicate completion (e.g., strikethrough or moved to a "Completed" section).

2.2.5 Delete Task

- **Description:** Users must be able to delete a task from their list.
- **Requirements:**
    - Users should be able to remove tasks by clicking a "Delete" button.
    - A confirmation dialog should appear before permanently deleting a task.

2.3 User Profile Management

2.3.1 Update Profile

- **Description:** Users must be able to update their personal information.
- **Requirements:**
    - Users should be able to update their name, email, and password.
    - The system should validate email format and check for uniqueness.

APPENDIX B: PROMPTS TO ENGINEERGPT

B.1.1 Prompt for Backend Development of CGPA & Transcript Application

Project Overview
Develop a comprehensive GPA calculator system for students that includes user authentication, course input for GPA calculation, and an admin interface for managing submitted transcript applications. The system should be robust and modular, integrating all functionalities into a single application.

Technology Stack
- Frontend: HTML5, CSS3, JavaScript, Bootstrap
- Backend: Node.js with Express.js
- Database: MySQL
- Authentication: JWT (JSON Web Tokens)

Provide mysql table creation queries in a separate file.

User Stories
For Students
1. Login:
o As a student, I want to log in using my credentials so that I can access my GPA calculation and transcript application functionalities.
2. Calculate GPA:
o As a student, I want to input my course details (course name, course code, credit hrs, marks, semester) so that I can calculate my GPA & CGPA. I shall be able to see the generated grade points and grades for each course as per BU grading policy.
o As a student, I want to see the calculated GPA after submitting my course details.
o As a Student, I shall be able import the required information about courses, credit hrs, marks, semester etc. in an excel file.
3. Submit Transcript Application:
o As a student, I want to submit an application for my transcript so that I can receive it from the exam department.

For Admins (Exam Department)
1. Login:
o As an admin, I want to log in using my credentials so that I can manage transcript applications.
2. View Applications:
o As an admin, I want to view all submitted transcript applications so that I can process them.
3. Update Application Status:
o As an admin, I want to update the status of a transcript application (Underprocess, Ready to Collect) so that students are informed about the progress.
4. Delete Application:
o As an admin, I want to delete an application so that I can manage the application records efficiently.

Functional Requirements
Authentication and Authorization
1. The system should support user registration and login using JWT for authentication.
2. Users should be able to log in using their username and password.
3. JWT tokens should be used to authenticate requests to the backend.

GPA Calculation
1. The GPA calculation page should allow students to input multiple courses with their respective grades and credits.
2. The system should calculate the GPA based on the input and display the result.
3. The calculation should convert letter grades to grade points using a predefined scale.

Transcript Application Submission
1. The system should allow students to submit their transcript applications.
2. The application form should include fields for student name, student ID, program, and contact information.
3. Submitted applications should be stored in the database for admin review.

Admin Interface
1. Admins should have a separate login interface.
2. Admins should be able to view all submitted transcript applications.
3. Admins should be able to update the status of an application.
4. Admins should be able to delete an application.

Non-Functional Requirements
1. Performance:
o The system should handle concurrent users efficiently.
2. Security:
o Sensitive data should be encrypted.
o Authentication should be secure and resilient against attacks.
3. Usability:
o The user interface should be intuitive and easy to use.
4. Scalability:
o The system should be scalable to accommodate future enhancements.

Grading Scale for Bahria University
Bahria University follows a specific grading scale to assign grade points to letter grades. Here is the grading scale:

Letter Grade	Grade Points	Marks
A	4.0	85
A-	3.7	80
B+	3.3	75
B	3.0	71
B-	2.7	68
C+	2.3	64
C	2.0	60
C-	1.7	57
D+	1.3	53
D	1.0	50
F	0.0	0-49

Develop the complete backend that will be used for the system.

B.1.2 Prompt for Admin Panel Front End Development of CGPA & Transcript Application

Project Overview
Develop a comprehensive GPA calculator system for students that includes user authentication, course input for GPA calculation, and an admin interface for managing submitted transcript applications. The system should be robust and modular, integrating all functionalities into a single application.

Technology Stack
• Frontend: HTML5, CSS3, JavaScript, Bootstrap

User Stories
For Students
1. Login:
o As a student, I want to log in using my credentials so that I can access my GPA calculation and transcript application functionalities.
2. Calculate GPA:
o As a student, I want to input my course details (course name, course code, credit hrs, marks, semester) so that I can calculate my GPA & CGPA. I shall be able to see the generated grade points and grades for each course as per BU grading policy.
o As a student, I want to see the calculated GPA after submitting my course details.
o As a Student, I shall be able import the required information about courses, credit hrs, marks, semester etc. in an excel file.
3. Submit Transcript Application:
o As a student, I want to submit an application for my transcript so that I can receive it from the exam department.

For Admins (Exam Department)
1. Login:
o As an admin, I want to log in using my credentials so that I can manage transcript applications.
2. View Applications:
o As an admin, I want to view all submitted transcript applications so that I can process them.
3. Update Application Status:
o As an admin, I want to update the status of a transcript application (Underprocess, Ready to Collect) so that students are informed about the progress.
4. Delete Application:
o As an admin, I want to delete an application so that I can manage the application records efficiently.

Functional Requirements
Authentication and Authorization
1. The system should support user registration and login using JWT for authentication.
2. Users should be able to log in using their username and password.
3. JWT tokens should be used to authenticate requests to the backend.
GPA Calculation

1. The GPA calculation page should allow students to input multiple courses with their respective grades and credits.
2. The system should calculate the GPA based on the input and display the result.
3. The calculation should convert letter grades to grade points using a predefined scale.

Transcript Application Submission

1. The system should allow students to submit their transcript applications.
2. The application form should include fields for student name, student ID, program, and contact information.
3. Submitted applications should be stored in the database for admin review.

Admin Interface

1. Admins should have a separate login interface.
2. Admins should be able to view all submitted transcript applications.
3. Admins should be able to update the status of an application.
4. Admins should be able to delete an application.

Non-Functional Requirements

1. Performance:
o The system should handle concurrent users efficiently.
2. Security:
o Sensitive data should be encrypted.
o Authentication should be secure and resilient against attacks.
3. Usability:
o The user interface should be intuitive and easy to use.
4. Scalability:
o The system should be scalable to accommodate future enhancements.

Grading Scale for Bahria University

Bahria University follows a specific grading scale to assign grade points to letter grades. Here is the grading scale:

Letter Grade	Grade Points	Marks
A	4.0	85
A-	3.7	80
B+	3.3	75
B	3.0	71
B-	2.7	68
C+	2.3	64
C	2.0	60
C-	1.7	57
D+	1.3	53
D	1.0	50
F	0.0	0-49

Backend is already developed and deployed.

Develop the frontend application for admin only as the student portal is already developed.

Backend Endpoints:
Auth/Users:
http://localhost:3000/api/users

```
router.post('/register', register);
router.post('/login', login);

http://localhost:3000/api/admin'
router.get('/applications', authMiddleware, viewApplications);
router.put('/applications/status', authMiddleware, updateApplicationStatus);
router.delete('/applications', authMiddleware, deleteApplication);
```

```
Database Structure:
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role ENUM('student', 'admin') NOT NULL
);

CREATE TABLE courses (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    course_name VARCHAR(100),
    course_code VARCHAR(50),
    credit_hours INT,
    marks INT,
    semester VARCHAR(20),
    FOREIGN KEY (user_id) REFERENCES users(id)
);

CREATE TABLE transcripts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    student_name VARCHAR(100),
    student_id VARCHAR(50),
    program VARCHAR(100),
    contact_info VARCHAR(100),
    status ENUM('Underprocess', 'Ready to Collect') DEFAULT 'Underprocess',
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

B.1.3 Prompt for Student Panel Front End Development of CGPA & Transcript Application

Project Overview
Develop a comprehensive GPA calculator system for students that includes user authentication, course input for GPA calculation, and an admin interface for managing submitted transcript applications. The system should be robust and modular, integrating all functionalities into a single application.

Technology Stack
• Frontend: HTML5, CSS3, JavaScript, Bootstrap

User Stories
For Students
1. Login:
o As a student, I want to log in using my credentials so that I can access my GPA calculation and transcript application functionalities.
2. Calculate GPA:
o As a student, I want to input my course details (course name, course code, credit hrs, marks, semester) so that I can calculate my GPA & CGPA. I shall be able to see the generated grade points and grades for each course as per BU grading policy.
o As a student, I want to see the calculated GPA after submitting my course details.
o As a Student, I shall be able import the required information about courses, credit hrs, marks, semester etc. in an excel file.
3. Submit Transcript Application:
o As a student, I want to submit an application for my transcript so that I can receive it from the exam department.
For Admins (Exam Department)
1. Login:
o As an admin, I want to log in using my credentials so that I can manage transcript applications.
2. View Applications:
o As an admin, I want to view all submitted transcript applications so that I can process them.
3. Update Application Status:
o As an admin, I want to update the status of a transcript application (Underprocess, Ready to Collect) so that students are informed about the progress.
4. Delete Application:
o As an admin, I want to delete an application so that I can manage the application records efficiently.

Functional Requirements
Authentication and Authorization
1. The system should support user registration and login using JWT for authentication.
2. Users should be able to log in using their username and password.
3. JWT tokens should be used to authenticate requests to the backend.
GPA Calculation
1. The GPA calculation page should allow students to input multiple courses with their respective grades and credits.

2. The system should calculate the GPA based on the input and display the result.
3. The calculation should convert letter grades to grade points using a predefined scale.

Transcript Application Submission

1. The system should allow students to submit their transcript applications.
2. The application form should include fields for student name, student ID, program, and contact information.
3. Submitted applications should be stored in the database for admin review.

Admin Interface

1. Admins should have a separate login interface.
2. Admins should be able to view all submitted transcript applications.
3. Admins should be able to update the status of an application.
4. Admins should be able to delete an application.

Non-Functional Requirements

1. Performance:
   - o The system should handle concurrent users efficiently.
2. Security:
   - o Sensitive data should be encrypted.
   - o Authentication should be secure and resilient against attacks.
3. Usability:
   - o The user interface should be intuitive and easy to use.
4. Scalability:
   - o The system should be scalable to accommodate future enhancements.

Grading Scale for Bahria University

Bahria University follows a specific grading scale to assign grade points to letter grades. Here is the grading scale:

Letter Grade	Grade Points	Marks
A	4.0	85
A-	3.7	80
B+	3.3	75
B	3.0	71
B-	2.7	68
C+	2.3	64
C	2.0	60
C-	1.7	57
D+	1.3	53
D	1.0	50
F	0.0	0-49

Backend is already developed and deployed.

Develop the frontend for the student only, we will develop frontend for admin separately.

UI must be excellent.

Backend Endpoints:

Auth/Users:

http://localhost:3000/api/users

```
router.post('/register', register);
router.post('/login', login);

http://localhost:3000/api/gpa'
router.post('/calculate', authMiddleware, calculateGPA);
const calculateGPA = async (req, res) => {
  const { courses } = req.body;
  const userId = req.user.id;

  try {
    for (const course of courses) {
      await   createCourse(userId,   course.courseName,   course.courseCode,
course.creditHours, course.marks, course.semester);
    }

    const userCourses = await getCoursesByUserId(userId);
    const gpa = calculateGPAFromCourses(userCourses);
    res.json({ gpa });
  } catch (error) {
    res.status(500).json({ message: 'Error calculating GPA', error });
  }
};

const calculateGPAFromCourses = (courses) => {
  let totalPoints = 0;
  let totalCredits = 0;

  for (const course of courses) {
    const gradePoint = getGradePoint(course.marks);
    totalPoints += gradePoint * course.creditHours;
    totalCredits += course.creditHours;
  }

  return totalPoints / totalCredits;
};

const getGradePoint = (marks) => {
  if (marks >= 85) return 4.0;
  if (marks >= 80) return 3.7;
  if (marks >= 75) return 3.3;
  if (marks >= 71) return 3.0;
  if (marks >= 68) return 2.7;
  if (marks >= 64) return 2.3;
  if (marks >= 60) return 2.0;
  if (marks >= 57) return 1.7;
  if (marks >= 53) return 1.3;
  if (marks >= 50) return 1.0;
  return 0.0;
};

http://localhost:3000/api/transcripts'
router.post('/submit', authMiddleware, submitTranscriptApplication);
const submitTranscriptApplication = async (req, res) => {
  const { studentName, studentId, program, contactInfo } = req.body;
  const userId = req.user.id;

  try {
    const   transcriptId   =   await   createTranscript(userId,   studentName,
studentId, program, contactInfo);
    res.status(201).json({ transcriptId });
  } catch (error) {
    res.status(500).json({    message:    'Error    submitting    transcript
application', error });
  }
};

http://localhost:3000/api/admin'
router.get('/applications', authMiddleware, viewApplications);
router.put('/applications/status',                          authMiddleware,
updateApplicationStatus);
router.delete('/applications', authMiddleware, deleteApplication);
```

Database Structure:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role ENUM('student', 'admin') NOT NULL
);

CREATE TABLE courses (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    course_name VARCHAR(100),
    course_code VARCHAR(50),
    credit_hours INT,
    marks INT,
    semester VARCHAR(20),
    FOREIGN KEY (user_id) REFERENCES users(id)
);

CREATE TABLE transcripts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    student_name VARCHAR(100),
    student_id VARCHAR(50),
    program VARCHAR(100),
    contact_info VARCHAR(100),
    status ENUM('Underprocess', 'Ready to Collect') DEFAULT 'Underprocess',
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

B.2.1 Prompt for Backend Development of To-Do List Application

1. Project Overview

The objective of this project is to develop a To-Do List application that allows users to create, manage, and organize their tasks efficiently. The system will include essential user authentication features like sign up, login, and logout, ensuring secure access to user-specific tasks. Each user will have their personalized to-do list, with options to add, edit, mark tasks as completed, and delete tasks.

Technology Stack

- Backend: Node.js with Express.js
- Database: MySQL
- Authentication: JWT (JSON Web Tokens)

Provide mysql table creation queries in a separate file.

2. Functional Requirements

2.1 User Authentication

2.1.1 Signup

Description: Users must be able to register for an account using their email and password.

Requirements:

a. A registration form must be provided for users to input their details (e.g., name, email, password).
b. Email must be unique for each user.
c. Passwords must meet security criteria (minimum length, use of alphanumeric characters).
d. Provide validation for all form fields (e.g., valid email format, non-empty fields).

2.1.2 Login

Description: Users must be able to log in with valid credentials.

Requirements:

a. A login form must be provided for users to enter their email and password.
b. Passwords must be hashed before being stored in the database.
c. Upon successful login, users should be redirected to their personal dashboard.
d. Invalid credentials should result in appropriate error messages.

2.1.3 Logout

Description: Users should be able to log out of the application securely.

Requirements:

a. Users should have an option to log out.
b. Session or JWT tokens should be invalidated on logout.

2.2 Task Management

2.2.1 Add Task

Description: Users must be able to add tasks to their to-do list.

Requirements:

a. Users should see a form or input field where they can enter task details (e.g., task title, due date, priority).
b. Tasks must be associated with the logged-in user.
c. The system should allow optional fields for due date and priority level (e.g., High, Medium, Low).

2.2.2 View Tasks

Description: Users must be able to view all of their tasks in a structured manner.

Requirements:

a. Tasks should be displayed in a list or table format.

b. Tasks can be filtered by status (e.g., pending, completed) or sorted by due date, priority, etc.
c. Completed tasks should be visually distinguishable from pending tasks.

2.2.3 Edit Task

Description: Users must be able to edit existing tasks.
Requirements:

a. Users should have the option to modify the task title, due date, or priority level.
b. Changes should be saved and reflected immediately.

2.2.4 Mark Task as Completed

Description: Users must be able to mark a task as completed.
Requirements:

a. Users should be able to click a button or checkbox to mark a task as completed.
b. The task should visually update to indicate completion (e.g., strikethrough or moved to a "Completed" section).

2.2.5 Delete Task

Description: Users must be able to delete a task from their list.
Requirements:

a. Users should be able to remove tasks by clicking a "Delete" button.
b. A confirmation dialog should appear before permanently deleting a task.

2.3 User Profile Management

2.3.1 Update Profile

Description: Users must be able to update their personal information.
Requirements:

a. Users should be able to update their name, email, and password.
b. The system should validate email format and check for uniqueness.
c. Develop the complete backend that will be used for the system.

B.2.2 Prompt for FrontEnd Development of To-Do List Application

1. Project Overview
The objective of this project is to develop a To-Do List application that allows users to create, manage, and organize their tasks efficiently. The system will include essential user authentication features like sign up, login, and logout, ensuring secure access to user-specific tasks. Each user will have their personalized to-do list, with options to add, edit, mark tasks as completed, and delete tasks.

Technology Stack
- Frontend: HTML5, CSS3, JavaScript, Bootstrap

2. Functional Requirements
2.1 User Authentication
2.1.1 Signup
Description: Users must be able to register for an account using their email and password.
Requirements:
a. A registration form must be provided for users to input their details (e.g., name, email, password).
b. Email must be unique for each user.
c. Passwords must meet security criteria (minimum length, use of alphanumeric characters).
d. Provide validation for all form fields (e.g., valid email format, non-empty fields).

2.1.2 Login
Description: Users must be able to log in with valid credentials.
Requirements:
a. A login form must be provided for users to enter their email and password.
b. Passwords must be hashed before being stored in the database.
c. Upon successful login, users should be redirected to their personal dashboard.
d. Invalid credentials should result in appropriate error messages.

2.1.3 Logout
Description: Users should be able to log out of the application securely.
Requirements:
a. Users should have an option to log out.
b. Session or JWT tokens should be invalidated on logout.

2.2 Task Management
2.2.1 Add Task
Description: Users must be able to add tasks to their to-do list.
Requirements:
a. Users should see a form or input field where they can enter task details (e.g., task title, due date, priority).
b. Tasks must be associated with the logged-in user.
c. The system should allow optional fields for due date and priority level (e.g., High, Medium, Low).

2.2.2 View Tasks
Description: Users must be able to view all of their tasks in a structured manner.
Requirements:
a. Tasks should be displayed in a list or table format.

b. Tasks can be filtered by status (e.g., pending, completed) or sorted by due date, priority, etc.
c. Completed tasks should be visually distinguishable from pending tasks.

2.2.3 Edit Task

Description: Users must be able to edit existing tasks.
Requirements:

a. Users should have the option to modify the task title, due date, or priority level.
b. Changes should be saved and reflected immediately.

2.2.4 Mark Task as Completed

Description: Users must be able to mark a task as completed.
Requirements:

a. Users should be able to click a button or checkbox to mark a task as completed.
b. The task should visually update to indicate completion (e.g., strikethrough or moved to a "Completed" section).

2.2.5 Delete Task

Description: Users must be able to delete a task from their list.
Requirements:

a. Users should be able to remove tasks by clicking a "Delete" button.
b. A confirmation dialog should appear before permanently deleting a task.

2.3 User Profile Management

2.3.1 Update Profile

Description: Users must be able to update their personal information.
Requirements:

a. Users should be able to update their name, email, and password.
b. The system should validate email format and check for uniqueness.

Backend is already developed and deployed.

Develop the frontend using following backend endpoints, and remember to develop separate pages for ease of usability and customization:

Backend Endpoints:

Auth/Users:

```
http://localhost:3000/api/users/signup
router.post('/signup', validateSignup, UserController.signup);
http://localhost:3000/api/users/login
router.post('/login', validateLogin, UserController.login);
http://localhost:3000/api/users/profile
router.put('/profile', authMiddleware, UserController.updateProfile);
```

Tasks:

```
http://localhost:3000/api/tasks/add
router.post('/add', authMiddleware, TaskController.addTask);
http://localhost:3000/api/tasks/view
router.get('/view', authMiddleware, TaskController.viewTasks);
http://localhost:3000/api/tasks/edit
router.put('/edit', authMiddleware, TaskController.editTask);
http://localhost:3000/api/tasks/completed
router.patch('/completed', authMiddleware, TaskController.completeTask);
http://localhost:3000/api/tasks/delete
router.delete('/delete', authMiddleware, TaskController.deleteTask);
```

Database Structure:
```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE tasks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    title VARCHAR(255) NOT NULL,
    due_date DATE,
    priority ENUM('High', 'Medium', 'Low'),
    status ENUM('Pending', 'Completed') DEFAULT 'Pending',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

APPENDIX C: GENERATED TEST CASES

C.1 Human Developed Test Cases for CGPA & Transcript Application

Test Case 1: Student Login

- **Test Case ID:** TC_001
- **Description:** Verify that a student can log in with valid credentials.
- **Preconditions:** Student account is created and activated.
- **Test Steps:**
    1. Open the login page.
    2. Enter a valid enrollment number and password.
    3. Click the "Login" button.
- **Expected Result:** Student should be redirected to the Dashboard page.
- **Postcondition:** Student is logged in successfully.

Test Case 2: Invalid Student Login

- **Test Case ID:** TC_002
- **Description:** Verify that a student cannot log in with invalid credentials.
- **Preconditions:** Student account exists.
- **Test Steps:**
    1. Open the login page.
    2. Enter an invalid username or password.
    3. Click the "Login" button.
- **Expected Result:** An error message should be displayed indicating incorrect credentials.
- **Postcondition:** Student remains on the login page.

Test Case 3: GPA Calculation – Manual Entry

- **Test Case ID:** TC_003
- **Description:** Verify GPA calculation for added courses.
- **Preconditions:** Student is logged in.
- **Test Steps:**
    1. Navigate to the GPA calculation page.
    2. Enter course details (e.g., Semester, course name, course code, credit hours, and marks).
    3. Add all courses of current semester in the given table
    4. Click the "Submit" button.
- **Expected Result:** The system should display the GPA, SGPA, and CGPA based on the added courses.
- **Postcondition:** GPA, SGPA, and CGPA are displayed correctly for the added courses.

Test Case 4: GPA Calculation – Import CSV

- **Test Case ID:** TC_004
- **Description:** Verify GPA calculation for imported courses.
- **Preconditions:** Student is logged in.
- **Test Steps:**
    1. Navigate to the GPA calculation page.
    2. Download file format
    3. Enter course details (e.g., Semester, course name, course code, credit hours, and marks) in CSV file.
    4. Upload CSV.

- **Expected Result:** The system should display the GPA, SGPA, and CGPA based on the imported courses.
- **Postcondition:** GPA, SGPA, and CGPA are displayed correctly for the added courses.

Test Case 5: Transcript Application Submission

- **Test Case ID:** TC_005
- **Description:** Verify that a student can submit a transcript application.
- **Preconditions:** Student is logged in.
- **Test Steps:**
    1. Navigate to the transcript application page.
    2. Fill out the application form (e.g., Transcript Type, Email, Mobile, and CNIC).
    3. Click the "Submit" button.
- **Expected Result:** The system should confirm the submission and store the application in the database.
- **Postcondition:** Transcript application is submitted and stored successfully.

Test Case 6: View Submitted Transcript Applications (Student)

- **Test Case ID:** TC_006
- **Description:** Verify that a student can view all its submitted transcript applications.
- **Preconditions:** Student is logged in, and there are submitted applications.
- **Test Steps:**
    1. Navigate to the transcript request page.
    2. View the list of submitted applications.
- **Expected Result:** The system should display all submitted transcript applications with relevant details.
- **Postcondition:** Student can view and access all submitted applications.

Test Case 7: Admin Login

- **Test Case ID:** TC_007
- **Description:** Verify that an admin can log in with valid credentials.
- **Preconditions:** Admin account is created and activated.
- **Test Steps:**
    1. Open the admin login page.
    2. Enter a valid email and password.
    3. Click the "Login" button.
- **Expected Result:** Admin should be redirected to the Dashboard page.
- **Postcondition:** Admin is logged in successfully.

Test Case 8: View Submitted Transcript Applications (Admin)

- **Test Case ID:** TC_008
- **Description:** Verify that an admin can view all submitted transcript applications.
- **Preconditions:** Admin is logged in, and there are submitted applications.
- **Test Steps:**
    1. Navigate to the transcript request page.
    2. View the list of submitted applications.
- **Expected Result:** The system should display all submitted transcript applications with relevant details.
- **Postcondition:** Admin can view and access all submitted applications.

Test Case 9: Update Transcript Application Status (Admin)

- **Test Case ID:** TC_009
- **Description:** Verify that an admin can update the status of a transcript application.

- **Preconditions:** Admin is logged in, and there are submitted applications.
- **Test Steps:**
    1. Navigate to the transcript management interface.
    2. Select an application and update its status (e.g., from "Underprocess" to "Ready to Collect" OR "Complete").
- **Expected Result:** The system should update the status and notify the student accordingly.
- **Postcondition:** Transcript application status is updated successfully.

Test Case 10: Delete Transcript Application (Admin)

- **Test Case ID:** TC_010
- **Description:** Verify that an admin can delete a transcript application.
- **Preconditions:** Admin is logged in, and there are submitted applications.
- **Test Steps:**
    1. Navigate to the transcript request page.
    2. Select an application and click the "Delete" button.
- **Expected Result:** The system should delete the application from the database.
- **Postcondition:** Transcript application is deleted successfully.

C.2 AI Developed Test Cases for CGPA & Transcript Application
Test Cases for Students

1. Student Login
Test Case ID: STU_LOGIN_01
Description: Verify that a student can log in using valid credentials.
Preconditions: The student must be registered in the system.
Steps:
1. Navigate to the student login page.
2. Enter valid username and password.
3. Click on the "Login" button.
Expected Result: The student should be redirected to the dashboard.

Test Case ID: STU_LOGIN_02
Description: Verify that a student cannot log in using invalid credentials.
Preconditions: The student must be registered in the system.
Steps:
1. Navigate to the student login page.
2. Enter invalid username and/or password.
3. Click on the "Login" button.
Expected Result: An error message should be displayed indicating invalid credentials.

2. Calculate GPA
Test Case ID: STU_GPA_01
Description: Verify that a student can input course details and calculate GPA.
Preconditions: The student must be logged in.
Steps:
1. Navigate to the GPA calculation page.
2. Input course details (course name, course code, credit hours, marks, semester).
3. Click on the "Calculate GPA" button.
Expected Result: The GPA should be calculated and displayed along with grade points and grades for each course.

Test Case ID: STU_GPA_02
Description: Verify that the GPA calculation is accurate based on the predefined grading scale.
Preconditions: The student must be logged in.
Steps:
1. Navigate to the GPA calculation page.
2. Input course details with known grades and credits.
3. Click on the "Calculate GPA" button.
Expected Result: The calculated GPA should match the expected GPA based on the predefined grading scale.

3. Import Course Details from Excel
Test Case ID: STU_IMPORT_01
Description: Verify that a student can import course details from an Excel file.
Preconditions: The student must be logged in.
Steps:
1. Navigate to the GPA calculation page.

2. Click on the "Import from Excel" button.
3. Select a valid Excel file with course details.
4. Click on the "Upload" button.
Expected Result: The course details should be imported and displayed on the GPA calculation page.

4. Submit Transcript Application
Test Case ID: STU_TRANSCRIPT_01
Description: Verify that a student can submit a transcript application.
Preconditions: The student must be logged in.
Steps:
1. Navigate to the transcript application page.
2. Fill in the application form (student name, student ID, program, contact information).
3. Click on the "Submit" button.
Expected Result: The application should be submitted and a confirmation message should be displayed.

Test Cases for Admins

1. Admin Login
Test Case ID: ADM_LOGIN_01
Description: Verify that an admin can log in using valid credentials.
Preconditions: The admin must be registered in the system.
Steps:
1. Navigate to the admin login page.
2. Enter valid username and password.
3. Click on the "Login" button.
Expected Result: The admin should be redirected to the admin dashboard.

Test Case ID: ADM_LOGIN_02
Description: Verify that an admin cannot log in using invalid credentials.
Preconditions: The admin must be registered in the system.
Steps:
1. Navigate to the admin login page.
2. Enter invalid username and/or password.
3. Click on the "Login" button.
Expected Result: An error message should be displayed indicating invalid credentials.

2. View Transcript Applications
Test Case ID: ADM_VIEW_01
Description: Verify that an admin can view all submitted transcript applications.
Preconditions: The admin must be logged in.
Steps:
1. Navigate to the transcript applications page.
Expected Result: A list of all submitted transcript applications should be displayed.

3. Update Application Status
Test Case ID: ADM_UPDATE_01
Description: Verify that an admin can update the status of a transcript application.

Preconditions: The admin must be logged in and there must be at least one submitted application.
Steps:
1. Navigate to the transcript applications page.
2. Select an application to update.
3. Change the status of the application (Underprocess, Ready to Collect).
4. Click on the "Update" button.
Expected Result: The status of the application should be updated and a confirmation message should be displayed.

4. Delete Application
Test Case ID: ADM_DELETE_01
Description: Verify that an admin can delete a transcript application.
Preconditions: The admin must be logged in and there must be at least one submitted application.
Steps:
1. Navigate to the transcript applications page.
2. Select an application to delete.
3. Click on the "Delete" button.
4. Confirm the deletion.
Expected Result: The application should be deleted and a confirmation message should be displayed.

C.3 Human Developed Test Cases for To-Do List Application

Test Case 1: User Signup

- **Test Case ID:** TC_001
- **Description:** Verify that a user can sign up with a valid email and password.
- **Preconditions:** User is on the signup page.
- **Test Steps:**
    1. Open the signup page.
    2. Enter a valid name, email, and password (meeting security criteria).
    3. Click the "Sign Up" button.
- **Expected Result:** The system creates a new account and displays a success message. The user is redirected to the login page.
- **Postcondition:** User account is successfully created.

Test Case 2: User Signup with Invalid Email

- **Test Case ID:** TC_002
- **Description:** Verify that the system displays an error for an invalid email format.
- **Preconditions:** User is on the signup page.
- **Test Steps:**
    1. Enter a valid name and password but an invalid email (e.g., "useremail.com").
    2. Click the "Sign Up" button.
- **Expected Result:** The system displays an error message: "Invalid email format."
- **Postcondition:** User account is not created.

Test Case 3: Duplicate Email Signup

- **Test Case ID:** TC_003
- **Description:** Verify that the system prevents signup with an already registered email.
- **Preconditions:** An account with the test email already exists.
- **Test Steps:**
    1. Enter an email that is already registered.
    2. Enter a valid name and password.
    3. Click the "Sign Up" button.
- **Expected Result:** The system displays an error message: "Email already exists."
- **Postcondition:** User account is not created.

Test Case 4: User Login

- **Test Case ID:** TC_004
- **Description:** Verify that a user can log in with valid credentials.
- **Preconditions:** User account is registered and activated.
- **Test Steps:**
    1. Open the login page.
    2. Enter a valid registered email and password.
    3. Click the "Login" button.
- **Expected Result:** The user is redirected to their personal dashboard.
- **Postcondition:** User is successfully logged in.

Test Case 5: User Login with Invalid Credentials

- **Test Case ID:** TC_005
- **Description:** Verify that login fails with incorrect credentials.
- **Preconditions:** User account exists.
- **Test Steps:**
    1. Open the login page.
    2. Enter an incorrect email or password.
    3. Click the "Login" button.

- **Expected Result:** The system displays an error message: "Invalid email or password."
- **Postcondition:** User is not logged in.

Test Case 6: User Logout

- **Test Case ID:** TC_006
- **Description:** Verify that a user can log out successfully.
- **Preconditions:** User is logged in.
- **Test Steps:**
    1. Click the "Logout" button in the application.
- **Expected Result:** The user is logged out, and the session is invalidated. The user is redirected to the login page.
- **Postcondition:** User session is terminated.

Test Case 7: Add New Task

- **Test Case ID:** TC_007
- **Description:** Verify that a user can add a new task to their to-do list.
- **Preconditions:** User is logged in.
- **Test Steps:**
    1. Navigate to the to-do list page.
    2. Enter a task title, optional due date, and priority.
    3. Click the "Add Task" button.
- **Expected Result:** The task is added to the to-do list and displayed on the page.
- **Postcondition:** Task is saved in the system.

Test Case 8: View All Tasks

- **Test Case ID:** TC_008
- **Description:** Verify that a user can view all of their tasks.
- **Preconditions:** User is logged in, and tasks have been added.
- **Test Steps:**
    1. Navigate to the to-do list page.
- **Expected Result:** All tasks are displayed in a list format, with completed tasks visually distinguishable from pending tasks.
- **Postcondition:** Tasks are displayed correctly.

Test Case 9: Edit an Existing Task

- **Test Case ID:** TC_009
- **Description:** Verify that a user can edit an existing task.
- **Preconditions:** User is logged in and has existing tasks.
- **Test Steps:**
    1. Locate a task and click the "Edit" button.
    2. Modify the task title, due date, or priority.
    3. Click the "Save" button.
- **Expected Result:** The task details are updated and reflected immediately in the task list.
- **Postcondition:** Task is updated successfully.

Test Case 10: Mark Task as Completed

- **Test Case ID:** TC_010
- **Description:** Verify that a user can mark a task as completed.
- **Preconditions:** User is logged in and has pending tasks.
- **Test Steps:**
    1. Locate a task and click the "Mark as Completed" checkbox or button.
- **Expected Result:** The task is marked as completed, visually updated (e.g., strikethrough or moved to a "Completed" section).

- **Postcondition:** Task status is updated to "Completed."

Test Case 11: Delete a Task

- **Test Case ID:** TC_011
- **Description:** Verify that a user can delete a task from their to-do list.
- **Preconditions:** User is logged in and has existing tasks.
- **Test Steps:**
    1. Locate a task and click the "Delete" button.
    2. Confirm the deletion in the dialog box.
- **Expected Result:** The task is removed from the list and deleted from the database.
- **Postcondition:** Task is successfully deleted.

Test Case 12: Update User Profile

- **Test Case ID:** TC_012
- **Description:** Verify that a user can update their profile information.
- **Preconditions:** User is logged in.
- **Test Steps:**
    1. Navigate to the profile page.
    2. Modify name, email, or password fields.
    3. Click the "Update Profile" button.
- **Expected Result:** The user's profile information is updated in the system.
- **Postcondition:** User profile is successfully updated.

Test Case 13: Validate Unique Email on Profile Update

- **Test Case ID:** TC_013
- **Description:** Verify that a user cannot update their profile with an email already registered by another user.
- **Preconditions:** Another user account exists with a different email.
- **Test Steps:**
    1. Navigate to the profile page.
    2. Attempt to update the email to one that already exists in the system.
    3. Click the "Update Profile" button.
- **Expected Result:** The system displays an error message: "Email already in use."
- **Postcondition:** Email is not updated.

C.4 AI Developed Test Cases for To-Do List Application

1. User Authentication

1.1 Signup

Test Case 1.1.1: Successful Signup
- **Description**: Verify that a user can successfully sign up with valid details.
- **Preconditions**: None
- **Steps**:
  1. Navigate to the signup page.
  2. Enter a valid name, email, and password.
  3. Click the "Sign Up" button.
- **Expected Result**: User is registered successfully and redirected to the login page.

Test Case 1.1.2: Duplicate Email Signup
- **Description**: Verify that a user cannot sign up with an email that is already registered.
- **Preconditions**: An account with the same email already exists.
- **Steps**:
  1. Navigate to the signup page.
  2. Enter a valid name, the duplicate email, and a password.
  3. Click the "Sign Up" button.
- **Expected Result**: An error message is displayed indicating that the email is already in use.

Test Case 1.1.3: Invalid Email Format
- **Description**: Verify that the system validates the email format during signup.
- **Preconditions**: None
- **Steps**:
  1. Navigate to the signup page.
  2. Enter a valid name, an invalid email format (e.g., "user@com"), and a password.
  3. Click the "Sign Up" button.
- **Expected Result**: An error message is displayed indicating that the email format is invalid.

Test Case 1.1.4: Password Security Criteria
- **Description**: Verify that the system enforces password security criteria during signup.
- **Preconditions**: None
- **Steps**:
  1. Navigate to the signup page.
  2. Enter a valid name, email, and a password that does not meet the security criteria (e.g., "123").
  3. Click the "Sign Up" button.
- **Expected Result**: An error message is displayed indicating that the password does not meet the security criteria.

1.2 Login

Test Case 1.2.1: Successful Login
- **Description**: Verify that a user can successfully log in with valid credentials.

- **Preconditions**: User account exists.
- **Steps**:
  1. Navigate to the login page.
  2. Enter a valid email and password.
  3. Click the "Login" button.
- **Expected Result**: User is logged in successfully and redirected to their personal dashboard.

Test Case 1.2.2: Invalid Credentials
- **Description**: Verify that the system displays an error message for invalid login credentials.
- **Preconditions**: User account exists.
- **Steps**:
  1. Navigate to the login page.
  2. Enter an invalid email or password.
  3. Click the "Login" button.
- **Expected Result**: An error message is displayed indicating invalid credentials.

1.3 Logout

Test Case 1.3.1: Successful Logout
- **Description**: Verify that a user can successfully log out.
- **Preconditions**: User is logged in.
- **Steps**:
  1. Click the "Logout" button.
- **Expected Result**: User is logged out successfully and redirected to the login page.

2. Task Management

2.1 Add Task

Test Case 2.1.1: Successful Task Addition
- **Description**: Verify that a user can successfully add a task.
- **Preconditions**: User is logged in.
- **Steps**:
  1. Navigate to the task addition form.
  2. Enter task details (e.g., title, due date, priority).
  3. Click the "Add Task" button.
- **Expected Result**: Task is added successfully and displayed in the task list.

2.2 View Tasks

Test Case 2.2.1: View All Tasks
- **Description**: Verify that a user can view all their tasks.
- **Preconditions**: User is logged in and has tasks.
- **Steps**:
  1. Navigate to the task list page.
- **Expected Result**: All tasks are displayed in a list or table format.

Test Case 2.2.2: Filter Tasks by Status

- **Description**: Verify that a user can filter tasks by status (e.g., pending, completed).
- **Preconditions**: User is logged in and has tasks with different statuses.
- **Steps**:
  1. Navigate to the task list page.
  2. Select a status filter (e.g., "Pending").
- **Expected Result**: Only tasks with the selected status are displayed.

2.3 Edit Task

Test Case 2.3.1: Successful Task Edit
- **Description**: Verify that a user can successfully edit a task.
- **Preconditions**: User is logged in and has tasks.
- **Steps**:
  1. Navigate to the task list page.
  2. Click the "Edit" button for a task.
  3. Modify the task details (e.g., title, due date, priority).
  4. Click the "Save" button.
- **Expected Result**: Task is updated successfully, and changes are reflected immediately.

2.4 Mark Task as Completed

Test Case 2.4.1: Successful Task Completion
- **Description**: Verify that a user can mark a task as completed.
- **Preconditions**: User is logged in and has tasks.
- **Steps**:
  1. Navigate to the task list page.
  2. Click the "Mark as Completed" button or checkbox for a task.
- **Expected Result**: Task is marked as completed and visually updated (e.g., strikethrough or moved to "Completed" section).

2.5 Delete Task

Test Case 2.5.1: Successful Task Deletion
- **Description**: Verify that a user can delete a task.
- **Preconditions**: User is logged in and has tasks.
- **Steps**:
  1. Navigate to the task list page.
  2. Click the "Delete" button for a task.
  3. Confirm the deletion in the confirmation dialog.
- **Expected Result**: Task is deleted successfully and removed from the task list.

3. User Profile Management

3.1 Update Profile

Test Case 3.1.1: Successful Profile Update
- **Description**: Verify that a user can update their profile information.
- **Preconditions**: User is logged in.
- **Steps**:
  1. Navigate to the profile update page.

  2. Modify the profile details (e.g., name, email, password).
  3. Click the "Save" button.
- **Expected Result**: Profile information is updated successfully.

Test Case 3.1.2: Email Uniqueness Check
- **Description**: Verify that the system checks for email uniqueness during profile update.
- **Preconditions**: Another user account exists with the same email.
- **Steps**:
  1. Navigate to the profile update page.
  2. Enter an email that is already registered to another user.
  3. Click the "Save" button.
- **Expected Result**: An error message is displayed indicating that the email is already in use.

Test Case 3.1.3: Invalid Email Format
- **Description**: Verify that the system validates the email format during profile update.
- **Preconditions**: User is logged in.
- **Steps**:
  1. Navigate to the profile update page.
  2. Enter an invalid email format (e.g., "user@com").
  3. Click the "Save" button.
- **Expected Result**: An error message is displayed indicating that the email format is invalid.