NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
DIVISION: COMPUTER SCIENCE

Statistical Analysis of AI Use in "Software Engineering" course projects by senior ECE students

Diploma Thesis

Of

CHRISTODOULOS STYLIANIDIS

Supervisors: Vassilios Vescoukis, Professor, NTUA

Athens, July, 2025

This page is left intentionally blank.

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
DIVISION: COMPUTER SCIENCE

Statistical Analysis of AI Use in "Software Engineering" course projects by senior ECE students

Diploma Thesis

Of

CHRISTODOULOS STYLIANIDIS

Supervisors: Vassilios Vescoukis, Professor, NTUA

Approved by the three-member scientific committee on 8th of July, 2025

Veskoukis Vasileios, Professor, NTUA	Nikolaos Papaspyrou, Professor, NTUA	Konstantinos Sagonas, Professor, NTUA

Athens, July, 2025

.....

Stylianidis Christodoulos

Graduate of School of Electrical and Computer Engineering, National Technical University of Athens

Περίληψη

Η εργασία εξετάζει πώς οι φοιτητές χρησιμοποιούν εργαλεία τεχνητής νοημοσύνης (AI) σε εργασίες ανάπτυξης λογισμικού, με βάση δεδομένα από τα μαθήματα «Τεχνολογία Λογισμικού» (7ο εξάμηνο) και «Τεχνολογίες Υπηρεσιών Λογισμικού» (8ο εξάμηνο) στη Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του ΕΜΠ. Αντί για ερωτηματολόγια, αναλύθηκαν περίπου 6.000 καταγραφές για το πότε, πώς, τι και γιατί οι φοιτητές χρησιμοποίησαν το AI και σε ποιό σημείο στον κύκλο ανάπτυξης. Κάθε εγγραφή περιλαμβάνει φάση, ενέργεια, γλώσσα προγραμματισμού, μοντέλο AI, επίπεδο εμπειρίας και υποκειμενικά αποτελέσματα (χρόνος, εξοικονόμηση, συναίσθημα, αίσθηση απειλής).

Τα αποτελέσματα δείχνουν ότι το AI είναι πιο αποδοτικό στις φάσεις κώδικα και δοκιμών, όπου τα παραγόμενα αποτελέσματα απαιτούν ελάχιστες διορθώσεις και εξοικονομούν σημαντικό χρόνο, συχνά διπλάσιο από τον αναμενόμενο. Στις αρχικές φάσεις, όπως η συλλογή απαιτήσεων ή ο σχεδιασμός αρχιτεκτονικής, η συμβολή είναι μικρότερη, καθώς το AI παρέχει γενικές ιδέες που χρειάζονται σημαντική προσαρμογή. Ένα δείγμα 51 φοιτητών συμμετείχε και στα δύο μαθήματα, επιτρέποντας συγκρίσεις εμπειρίας μεταξύ των εξαμήνων.

Η ανάλυση δείχνει διαφοροποιήσεις ανά γλώσσα και εμπειρία: Python και JavaScript συνδέονται με υψηλότερη ποιότητα, ενώ YAML και SQL εμφανίζονται σπάνια λόγω της φύσης τους. Οι λιγότερο έμπειροι φοιτητές βλέπουν το AI ως υποστηρικτή με μεγάλη χρήση, ενώ οι πιο έμπειροι το χρησιμοποιούν κριτικά. Συνολικά, η τεχνητή νοημοσύνη έχει εξελιχθεί από προαιρετικό βοήθημα σε αναπόσπαστο εργαλείο ανάπτυξης λογισμικού που απαιτεί τεχνική και εκπαίδευση.

Λέξεις Κλειδιά: τεχνητή νοημοσύνη, τεχνολογία λογισμικού, φοιτητές και AI, ανάλυση δεδομένων, υποβοήθηση κώδικα, ποιότητα λογισμικού, εξοικονόμηση χρόνου, chatGPT.

This page is left intentionally blank.

Abstract

This thesis explores how university students engage with AI tools during actual software development projects; it uses structured data collected from specific courses, namely "Software Engineering" (7th semester) and "Software as a Service Technologies" (8th semester), both in Curriculum Flow "L", at the school of Electrical and Computer Engineering, National Technical University of Athens. Rather than relying on post-project surveys, this study analyzes nearly 6,000 records submitted during the semester projects' execution, whichdocument when, how, what, and why students used AI during the entire software development cycle. Each submission includes metadata—such as the phase, scope, action, programming language and AI model used, declared experience level on action and tool use, and perceived outcomes in time allocation, time saved, feeling and "threat" feeling level.

Findings reveal that AI support is most effective in code-intensive phases like coding and testing, where students frequently describe outputs as usable with minimal revision or edit. Time savings are substantial, with many students reporting that AI completed tasks faster than they could on their own, at a ratio of at least double the time allocated. In contrast, conceptual software development phases like requirement gathering and architecture benefit more modestly—AI helps by only offering generic ideas which often lack specificity or contextual relevance resulting in need for major changes, or even characterized as unusable by the user. A subset of 51 students were registered in both courses "Software Engineering" and "Software as a Service Technologies Technologies", which enabled the analysis of the students' experience level in the first course in comparison to the second course.

The study also highlights important differences based on programming language and user experience. Python and JavaScript are associated with the highest perceived quality, while configuration or declarative languages like YAML or SQL tend to be mentioned in fewer records, indicating that the students haven't used much AI for those languages, which partially relates to the nature of the semester projects, focusing less on SQL databases and more on architecture and services. Inexperienced users tend to view AI as a helpful partner, while experienced students approach it more critically, balancing convenience with caution. These insights suggest that AI has moved from optional add-on to a "sine qua non" tool in software development—one that requires not only technical training but also reflective guidance.

Keywords: AI in software engineering, student-AI interaction, analysis over time, AI-assisted coding, language model effectiveness, code quality, time savings through AI tools, data analysis.

This page is left intentionally blank.

Acknowledgements

First and foremost, I would like to show my deepest gratitude to my Professor, Dr. Vassilios Vescoukis, for his constant guidance and insightful feedback throughout the development of this thesis.

Special thanks are due to PhD researcher, Mr. Christos Hadjichristofi, who helped me with some of the most major steps for the completion of this work. His assistance in navigating with unfamiliar tasks was invaluable. Without the support and contribution of both, this thesis would not have been possible.

I would like to acknowledge the examination committee for their role in evaluating this thesis. Their formal oversight ensures the academic standards of the program are upheld.

I am grateful to my fellow students for their willingness to review parts of my thesis and offer a fresh perspective when I had grown too close to the material. Their comments and observations helped me recognize blind spots and improve the understanding so that the new readers would receive the point I was attempting to pass.

I would also like to acknowledge the School of Electrical and Computer Engineering at the National Technical University of Athens for providing the academic environment and infrastructure necessary for carrying out this research. The resources, courses and faculty support through the entirety of my studies formed the foundation on which this thesis could be built.

Table of Contents

Table of Figures

0. Εκτενή Περίληψη

Η παρούσα διπλωματική εργασία εξετάζει, μέσω έρευνας, τη χρήση εργαλείων Τεχνητής Νοημοσύνης (AI) από φοιτητές του Εθνικού Μετσόβιου Πολυτεχνείου κατά την ανάπτυξη λογισμικού. Η μελέτη βασίζεται σε δεδομένα που συλλέχθηκαν από τα μαθήματα «Τεχνολογίας Λογισμικού» (7ο εξάμηνο) και «Τεχνολογίες Υπηρεσιών Λογισμικού» (8ο εξάμηνο) της Ροής Λ της ΣΗΜΜΥ. Αντί ερωτηματολογίων, χρησιμοποιήθηκαν περίπου 6.000 καταγραφές (JSON) που τεκμηριώνουν πραγματικές αλληλεπιδράσεις φοιτητών με εργαλεία AI σε όλες τις φάσεις του κύκλου ανάπτυξης λογισμικού.

Κάθε καταγραφή περιλαμβάνει φάση ανάπτυξης, ενέργεια, εύρος εφαρμογής, γλώσσα προγραμματισμού, εργαλείο και έκδοση AI, εκτιμώμενο χρόνο, εμπειρία χρήστη, ποιότητα αποτελέσματος, συναίσθημα και επίπεδο «απειλής». Η διαδικασία καθαρισμού και ελέγχου εγκυρότητας περιόρισε τα δεδομένα σε ~3.800 έγκυρες και ~1.000 «γκρίζες» εγγραφές, εξασφαλίζοντας αξιόπιστη στατιστική ανάλυση.

Τα αποτελέσματα δείχνουν ότι η τεχνητή νοημοσύνη χρησιμοποιείται αποδοτικότερα στις φάσεις συγγραφής κώδικα και testing. Εκεί, οι φοιτητές αξιολόγησαν το παραγόμενο αποτέλεσμα ως «έτοιμο προς χρήση» ή με «μικρές διορθώσεις» και ανέφεραν σημαντική εξοικονόμηση χρόνου — σε ποσοστά που αντιστοιχούν τουλάχιστον στο διπλάσιο του χρόνου που θα απαιτούσε η ίδια εργασία χωρίς AI. Αντίθετα, στις πρώιμες φάσεις όπως ανάλυση απαιτήσεων ή αρχιτεκτονικός σχεδιασμός, το AI παρείχε κυρίως γενικές ιδέες που χρειάζονταν σημαντική αναθεώρηση.

Η παρακολούθηση 51 φοιτητών που συμμετείχαν και στα δύο μαθήματα έδειξε σαφή αύξηση της αυτοπεποίθησης, της εμπειρίας και της ωριμότητας στη χρήση AI. Οι φοιτητές του δεύτερου μαθήματος δήλωσαν υψηλότερη ευχέρεια στη χρήση, καλύτερη αξιολόγηση των αποτελεσμάτων και πιο θετική στάση απέναντι στο AI, με σταθερή αισιοδοξία για τη μελλοντική του χρήση.

Η ανάλυση ανά γλώσσα προγραμματισμού έδειξε ότι η Python και η JavaScript συνδέονται με την υψηλότερη αντιληπτή ποιότητα, ενώ δηλωτικές ή περιγραφικές γλώσσες όπως YAML και SQL είχαν περιορισμένη παρουσία, κυρίως λόγω της φύσης των έργων. Οι λιγότερο έμπειροι φοιτητές αντιλαμβάνονται το AI ως συνεργάτη, ενώ οι πιο έμπειροι το αντιμετωπίζουν πιο κριτικά, γνωρίζοντας τα όριά του και τον κίνδυνο υπερεξάρτησης.

Συνολικά, η εργασία τεκμηριώνει ότι η τεχνητή νοημοσύνη έχει μετατραπεί από πειραματικό βοήθημα σε αναπόσπαστο στοιχείο της εκπαίδευσης και πρακτικής ανάπτυξης λογισμικού. Η συμβολή της χρήσης του είναι μέγιστη στις φάσεις παραγωγής κώδικα, σημαντική στη βελτίωση παραγωγικότητας και μάθησης, αλλά περιορισμένη σε εννοιολογικά στάδια όπου απαιτείται ερμηνευτική κρίση. Η μετάβαση αυτή καθιστά αναγκαία τη διδασκαλία της κριτικής χρήσης των εργαλείων AI και την ενσωμάτωσή τους στο πρόγραμμα σπουδών με παιδαγωγικό και όχι μηχανιστικό τρόπο.

1. Introduction

1.1 Context and Motivation

Artificial intelligence tools—especially those powered by large language models—are changing how software is built. What began as an experimental advantage for experts has become a regular part of software development routines. We focus on students enrolled in two Software Engineering courses in the School of Electrical and Computer Engineering of NTUA. In both courses, students need to implement non-trivial software development projects, in groups of 3 to 6. Probably, there are plenty of tasks that can be assisted by AI tools. Students have been instructed to use AI not just to write code, but also to conceive and document requirements, architectures, designs, testing etc.. These tools became part of everyday development activity, not just side experiments. Yet while the excitement around AI is clear, serious real-life data from real practitioners on how it fits, and how it behaves during the developmental cycle is missing. This thesis steps into that gap. It looks directly at how students used AI tools during their actual academic software projects, using a large, structured dataset collected during coursework of the two SE courses at the National Technical University of Athens.

1.2 Objective

The aim of this thesis is to examine how AI tools fit into all stages of student software development work—from early conceptual planning to hands-on coding and testing. The analysis focuses on how students evaluate the usefulness, threat, and overall impact of AI tools, and how their perceptions shift based on the type of task, the programming language, or their own level of experience. Instead of using general surveys or recollections, this study used real student-AI interaction logs. This makes it possible to study how AI actually supports or disrupts different types of software work in a developmental cycle setting.

1.3 Scope and Data Source

The data for this study comes from nearly 6,000 structured JSON files, submitted by students across the two software engineering courses—Software Engineering, Software as a Service Technologies—at the span of 3 semesters, thus collecting data from 3 different sets of students, Software Engineering during the years 2023 and 2024 and Software as a Service Technologies during 2024, marked accordingly as softeng23b, saas24a and softeng24b. Each submission, documents a single AI-assisted task: what kind of help the student asked for, what phase of development or pre-development they were in, what programming language they used—if any, how much time they allocated for the entire interaction and how

much time they estimate they saved along with how they felt about the result. Phases range from initial requirement gathering to deployment. Every entry includes detailed metadata—development phase, action performed, scope of the task, chosen AI tool, experience levels, time estimates, quality judgments, and emotional responses. This depth of structure allows for detailed analysis across time, tools, tasks, and user experience.

1.4 Methodological Overview

To make the dataset reliable, I analyzed all the combinations of choices per phase between action-scope-programming language and AI tool and labeled each path as valid or gray—where ambiguous paths were marked as. Based on those paths through the data mining tool Tableau Prep and python scripts for each phase and each dataset, a new column for each submission was created, displaying that submission's validity—valid, invalid or gray. For example, a student's submission who marked code management in the phase of requirements gathering would be marked as invalid, since clearly code management makes no sense being in the requirements phase. Only valid and gray entries were kept for analysis. Each diagram was meticulously created through Tableau Desktop, then each diagram's data was exported as a csv and displayed through an interactive web platform, where the readers can explore by course, task, user experience.

1.5 Contributions

This thesis explores how students actually use AI during the developmental cycle. Most submissions are focused on coding and testing phases, while it also equally examines non-code related phases like architecture and design. More submissions are reported in Python and JavaScript—also reporting better AI quality. Over time, students showed higher confidence, better judgment and more effective tool use.

1.6 Structure of the Thesis

The thesis is organized to build up the argument step by step. Chapter 2 introduces the research questions that shape the analysis. Chapter 3 outlines the constraints—what limited the data, and what that means for the results. Chapter 4 defines the dataset in full detail, including the dimensions and values captured in each course. Chapter 5 profiles the student population, analyzes the demographics and explains how experience was defined as. Chapter 6 explains the methodology, and all the additional fields created to aid the visualization. Chapter 7 presents the actual findings of each research question. Chapter 8 interprets those results in a broader context. Finally, Chapter 9 offers conclusions for the research. Also, in Appendix A we can see all the correct paths—used for validation, for each dataset—and in Appendix B two random sample entries from students.

2. Related Work

2.1 Novice Programmers' Patterns with Copilot [1]

Prior studies have explored the integration of AI tools such as GitHub Copilot in software engineering classes as an educational tool. The courses were undergraduate programming courses, and they examined how CS1 level students interacted with Copilot, identifying behavioral patterns such as passive acceptance of code produced and exploratory prompting. The study revealed both the perceived benefits of faster progress and the cognitive challenges students faced in understanding or verifying AI-generated code. Similarly they conducted a controlled experiment comparing performance with and without AI use in brownfield programming tasks. Their log-based analysis demonstrated significant time savings and reduced search activity, yet highlighted student concerns about over-reliance and comprehension loss. Many students found Copilot as "eerie" yet helpful ("its weird that it knows what I want"). This duality aligns with the present thesis's examination of both perceived benefits and risks in student-AI tool interaction.

2.2 "Give me the code": First-Year Student ChatGPT Logs [2]

This is the analysis of 69 first-year students' structured log files documenting their use of ChatGPT. These students documented their AI interactions with JSON-like logs. The data revealed distinct usage patterns such as single-shot code requests and iterative refinement through follow-up prompts. Despite limited training in prompt engineering, many students effectively incorporated AI-generated suggestions into their code. The study highlighted student agency in evaluating AI outputs and showed how structured logging enables fine-grained insights into prompting behavior and decision-making during development. This work aligns with the present thesis in both methodology (log-based tracking) and focus (AI use by novice developers during coursework)

2.3 Copilot in Brownfield Programming Tasks [3]

A controlled experiment was conducted to asses GitHub copilot's impact on undergraduate students performing brownfield programming tasks—modifying or extending legacy codebases. Ten participants were observed across Copilot-assisted and non-assisted conditions. Using detailed IDE event logs, the study measured metrics such as time-to-completion, code progress, and auxiliary search activity. Results showed a 35% improvement in completion speed and 50% increase in solution progress when Copilot was used. The tool also significantly reduced time spent on typing and external searches. Despite these gains, students raised concerns about understanding and trusting the AI-generated code, with several expressing unease about relying on suggestions they could not fully verify.

Like this thesis, the study avoids speculation and focuses on concrete usage data to examine how students interact with AI tools during real development work.

2.4 Youth Learners Using LLM Code Generators [4]

Kazemitabaar et al. (2023) explored how kids and teens, aged 10 to 17, used Codex to learn Python on their own. By logging every prompt and completion across 45 coding tasks, the researchers were able to see distinct usage styles—some students asked Codex to solve entire tasks in one go, others worked through problems step by step, and a few barely used the AI at all. The one-shot users often got quick results, but those who broke tasks down or mixed their own code with AI suggestions showed stronger understanding later on. Some students blindly accepted Codex's output, while others questioned it, tested it, or rewrote parts. This kind of variation echoes patterns found in the present thesis: experience level, prompting style, and task structure all shape how students use AI tools—and how much they actually learn from them. Both studies rely on concrete usage logs rather than self-reported impressions.

2.5 Exploring the Integration of ChatGPT in Education [5]

Elbanna and Armstrong examined how ChatGPT can be integrated into educational settings, highlighting its potential to streamline academic tasks and support personalized learning. They discussed both opportunities and limitations, emphasizing the need for critical adoption. Although broad in focus, the study recognizes shifts in student behavior and performance tied to AI use. Its relevance to this thesis lies in its framing of AI as an evolving academic tool that directly affects student workflows, particularly in task-based settings.

2.6 Assessing the Effectiveness of ChatGPT in Preparatory Testing Activities [6]

Haldar, Pierce, and Capretz conducted a focused study where students used ChatGPT to draft software test artifacts. The students compared AI-generated and manually written test plans, observing both time savings and gaps in test completeness. Though modest in scale, this work deals directly with how AI tools integrate into software development processes. Its attention to student workflow and output quality, based on artifact inspection, makes it directly relevant to this thesis.

2.7 A Review on the Perks of Using ChatGPT in Education [7]

Heathen and Lin provide a broad review of ChatGPT's role in education, cataloging reported benefits such as accessibility, feedback support, and creativity boosts. The study is descriptive and not empirically grounded, focusing more on theoretical affordances than

behavior data. While its scope includes computer science use cases, its lack of log-based or output-driven analysis limits direct comparison with the present thesis, though it supports contextual framing.

2.8 Response Strategies to ChatGPT in Language Education [8]

Liu et al. investigated the influence of ChatGPT on Chinese language education. The study focused on classroom practices, response behaviors, and adaptive strategies by instructors and students. It is strictly confined to language education and does not engage with programming or development tasks. Thus, while informative for broader educational use of AI, it does not intersect meaningfully with the thesis's software-oriented log-based analysis.

2.9 The Role of ChatGPT in Higher Education [9]

Rasul et al. outlined five key benefits and five risks of ChatGPT in higher education, touching on productivity, feedback, and integrity concerns. Their work synthesizes current literature and proposes future research directions. This paper reflects many of the motivations behind the thesis, improving efficiency without undermining understanding. The link to the current thesis lies in its thematic alignment around balancing automation with learning depth.

2.10 ChatGPT-Facilitated Programming Tasks [10]

Sun et al. explored how students used ChatGPT to assist with programming assignments in a college course. One group used ChatGPT freely, while another worked without AI. Logs showed that AI-assisted students engaged more in debugging and iterative refinement. Even though raw performance wasn't drastically different, the AI group reported higher confidence and motivation. The focus on concrete behavioral data and coding logs, rather than survey impressions, makes this study directly comparable to the present thesis.

2.11 Prompt-Based Learning with ChatGPT [11]

In a follow-up, Sun et al. trained students to use structured prompts when interacting with ChatGPT. Those with prompt training used the tool more strategically and showed deeper problem understanding. Logs revealed that structured prompting led to more relevant responses and better learning outcomes. Like this thesis, the study connects specific user behavior—captured through logs—to concrete educational gains in programming tasks.

2.12 Eight-Week Use of ChatGPT in Python Course [12]

Ma et al. integrated ChatGPT into a Python programming course and analyzed logs from student-AI conversations. Students used the tool for debugging, exploring variations, and clarifying concepts. Their feedback was mostly positive, especially regarding increased independence and reduced frustration. The study's reliance on structured usage data aligns with this thesis's focus on how students practically engage with AI in real coding scenarios.

2.13 Role of AI Literacy in Using ChatGPT for Code [13]

Wang et al. analyzed how students' AI literacy and programming experience affected their ability to use ChatGPT for coding problems. Log data showed that those with more experience and critical understanding used the tool more effectively and caught more errors. This mirrors the current thesis in examining how background factors influence AI tool use, based on detailed interaction traces.

2.14 AI in Preparatory Software Testing [14]

Haldar et al. studied graduate students using ChatGPT to generate software test plans and cases. Students reviewed and modified the AI-generated output, integrating it into their workflow. Although ChatGPT saved time, students noted that its output wasn't complete without human judgment. The process-based analysis of how AI fits into real development phases makes this paper methodologically like the present thesis.

2.15 ChatGPT on Software Testing Course Questions [15]

Jalil et al. tested ChatGPT's ability to answer structured questions from a software testing course. They evaluated both answers and explanations, finding moderate accuracy and several logic flaws despite confident tone. Though not a student usage study, it contribute [5]s to understanding AI's limitations in software contexts. This aligns with the current thesis in examining task-level performance through structured, content-based evaluation.

2.16 ChatGPT in Undergraduate Data Structures Course [16]

Qureshi ran a classroom experiment where students used either traditional methods or ChatGPT to solve data structure problems. The AI group performed better overall but sometimes submitted code with subtle errors. The study focused on outcome metrics and code accuracy, not just perceptions, which is consistent with the thesis's log-based evaluation of AI effectiveness in academic programming.

3. Research Questions

The research questions were selected in order to answer how AI tools are actually used across different phases of the developmental cycle; to make sure that each dataset is reliable; evaluate how helpful AI is in pre-code related and code related phases; how the students' use of AI changes over time; and to analyze AI responds in behavior and output quality per coding language used.

3.1 Phase Categorization

The software development lifecycle is split into two major categories: pre-code related phases and code related phases. Pre-code related phases include all planning and design-related activities that don't involve direct code execution. These include phases like requirements gathering, requirements specification, system architecture, and design modeling. They are typically abstract, text-driven, and involve activities such as stakeholder analysis, documentation, and UML-based planning. In other cases they are detailed explanations of methods for how the databases could be distributed or how the developer needs to follow a specific set of rules in order to maintain the fundamental requirements needed for the project.

The code related phases, by contrast, include coding, testing, and deployment. These are the stages where actual source code is written, tested, and deployed into operational environments. Code mainly includes source code authoring and code management for the frontend, backend and api. Testing is further broken into unit testing, functional testing, integration testing, and performance testing, where the peak choices were functional testing and unit testing. Even though submissions aren't that many comparing to coding and testing, deployment involves mostly dev-ops and deployment scripts-related work. This division between abstract planning, meaning non-code related phases and concrete execution, so, code related phases, forms the basis for comparing AI support for the entire developmental cycle of a software development.

3.2 Source of Data

The dataset used for this analysis was collected from two advanced software engineering courses—Software Engineering and Software as a Service Technologies—at the span of 3 semesters, thus collecting data from softeng23b saas24a and softeng24b, which were offered at the National Technical University of Athens. As part of course requirements, students submitted structured .json files describing AI-assisted tasks they completed during the entirety of the software development cycle. Each submission captured the phase, the specific action performed, the scope of the task, the programming language used—if any—the AI tool and version used, and self-declared experience for both tool and action type. It also included self-reported fields on time allocated, time saved, output quality of AI, emotional stance, and perceived threat from AI involvement. All these dimensions were selected from a pre-existing set of fields, reducing ambiguity from free user choices.

Some differences existed across courses: for example, softeng23b used categorical descriptors like “none,” “little,” or “big” for experience levels, while saas24a and softeng24b used numeric scales from 0 to 5 where 0 is the least experience and 5 is master. Additionally, not all courses covered the same development phases—saas24a excluded requirements gathering, requirements specification and testing, while the others included all seven. These discrepancies were resolved during preprocessing by harmonizing the schema across datasets. Invalid or structurally inconsistent entries were removed or flagged as gray, resulting in a clean, normalized dataset suitable for comparative and cross-phase analysis.

3.3 Differences across datasets

The tables below show the different choices available to a student for each column that had a different choice-set across all three datasets.

	SoftEng-23b	SaaS-24a	SoftEng-24b
action experience	none	0	0
	little	1	1
	fair	2	2
	big	3	3
		4	4
		5	5

	SoftEng-23b	SaaS-24a	SoftEng-24b
tool experience	none	0	0
	some	1	1
	enough	2	2
	master	3	3
		4	4
		5	5

	SoftEng-23b	SaaS-24a	SoftEng-24b
quality of ai use	unusable	0	0
	major modifications needed	1	1
	minor modifications needed	2	2
	ready-to-use	3	3
		4	4
		5	5

generic feeling, exists as generic feeling now and generic feeling future for the databases of SaaS-24a and SoftEng-24b, whereas SoftEng-23b only includes the one column of “generic feeling” as shown below

	SoftEng-23b	SaaS-24a	SoftEng-24b
generic feeling	makes not sense	0	0
	needs work	1	1
	great in the future	2	2
	great as-is	3	3
		4	4
		5	5

3.4 Research Questions

The focus of this thesis is built around six central research questions. Each question addresses a specific aspect of student–AI interaction: data quality and validity, phase-wise usage patterns, dependencies per language or experience of the student, and evolution of students’ AI use over time.

The validity of each submission was based on a schema for each databases’ phase, thus all the semantic and valid paths for all the combinations of action-scope-language and AI tool used, were found and used as a reference in order to mark each submission as valid-invalid or gray. The most ambiguous choices were where the student chose a borderline accepted choice and were marked as gray—since they weren’t invalid but not valid either. This is

mainly due to the student's uncertainty or lack of understanding regarding the available options given in phase-action-scope on the action they actually used the tool for. The level of experience for each user is selected based on their declared answer in tool and action experience; a student was marked as experienced if they had in both fields, values of 3 or higher—in the case of the categorical choices, "enough" or "some" and higher—otherwise they were marked as inexperienced users.

RQ0: Can student-submitted data be reliably used for analysis? This question tests the structural integrity and reliability of the dataset. It looks at how many submissions are valid, gray, or invalid, and whether data quality correlates with user experience, programming language, or year of study. This question ensures that the dataset is trustworthy before deeper analysis is performed.

RQ1: How do students use AI in pre-development (non-code related) phases? This question focuses on early conceptual phases—requirements gathering, requirements specification, architecture, and design. This question examines the frequency of AI use in these phases, the types of tasks for which AI tool was user and how the students evaluate the quality of AI output based on their experience level. It also analyzes how usage patterns differ by student experience and AI familiarity.

RQ2: How do students use AI in code-related development phases? This question targets coding, testing, and deployment. It examines how often AI is used in these stages, whether the generated content needs correction, how much time students say they save, and what feeling they have about AI now or in the future. It evaluates whether AI serves as a reliable coding partner or still demands close oversight.

RQ3: Does programming language affect AI-perceived usefulness? This question examines whether the effectiveness of AI tools varies depending on the programming language used. It tests if some languages—like Python or JavaScript—produce more usable outputs with better quality and as little modification needed as possible, while others—like YAML or SQL—introduce errors or mismatches. It also looks at how task types and language complexity interact to affect utility.

RQ4: How does student experience influence perception and effectiveness of AI? This question compares responses from inexperienced versus experienced users. It investigates whether the users who marked themselves as inexperienced report higher satisfaction, stronger trust and whether more experienced users are more skeptical or selective. It evaluates if AI serves different roles depending on how comfortable the user is with the tool and action.

RQ5: How does AI perception evolve over time in tracked users? This question follows a subset of students who appear in both softeng23b and saas24a. It tracks changes in tool and action confidence, emotional stance, and perceived value across semesters. It identifies whether repeated exposure to AI shifts behavior, strengthens critical judgment, or builds long-term trust in AI as a development resource.

Each research question is supported by structured metadata, normalized fields, and diagrammatic visualization. Together, they form a comprehensive framework for analyzing the multi-dimensional role of AI in software engineering's development lifecycle.

4. Constraints

The conditions under which these datasets are collected, cleaned, and interpreted created some limitations. This section explains the main limitations and how far its conclusions can be drawn as. These limitations are mainly created from four areas: who the participants were, how their submissions were filtered, how the logging system evolved across courses, and the broader academic setting in which everything took place.

4.1 Sample Constraints

All data analyzed in this thesis comes from students who enrolled in “Software Engineering” and “Software as a Service Technologies” during the academic years 2023-2024 and 2024-2025. These were taken by students—who had already taken classes where they each had the same training in programming, software design, and systems development—and not random people who use AI tools and LLM’s. Therefore, the dataset shows the behavior of students in a formal, engineering-focused academic environment, not learners from non-technical knowledge, professional developers who have experience, or self taught coders who have actual jobs outside academic frameworks.

Importantly, participation in the AI usage logging process was not optional—it was formally integrated into the grading structure of each course. This means that some students felt the need to submit their responses hastily without really paying attention to what they were submitting—resulting in multiple entries of the same responses including the exact same interaction with the AI model or random choosing that makes no semantical sense to be chosen with consciousness—in order to avoid the grade penalty if they didn’t submit their AI logs.

No self-reported demographic data—such as socioeconomic background, gender identity, or linguistic diversity—was collected. Besides enrollment year, which was derived from the student’s username and allows for approximate age estimation, no other field captures the broader social or cultural context. There is no information about prior AI usage habits, or more exposure than the average student in an academic setting. As a result, the conclusions drawn from this dataset reflect a specific and academically advanced student population, and any attempt to extrapolate these findings to a more broad developer population cannot be generalized without additional context.

4.2 Data Filtering Constraints

The raw data analyzed in this study was submitted by students in the form of .zip files, each recording included a .json file with the structural format given to the students for each course—and an optional .txt file including the entire student-AI interaction. Each .json file included one answer per column, and since in each course the choices were different, the template was given in the beginning of each course. Also, all columns were required in order to be accepted by the web platform and stored in the database. If the zip file included the AI interaction, the entire content of .txt was stored under the "prompt" column in the database.

Since not all submissions had semantic meaning in between combination choices, a filtering and validation process was applied to all incoming data. This matrix was defined per phase for each database differently and checked all the valid combinations for action-scope-programming language used and AI model. For example, we couldn't allow submissions during the coding phase who also answered "UML deployment" as scope, since it makes zero sense to actually be in the coding phase during UML diagrams. Some choices include gray coloring, which means that the specific set of combination is ambiguous either due to the misunderstanding of the student or the choice being borderline accepted in that phase. Below is an example of two phases—coding and deployment—in the dataset SaaS-24a where we can clearly see the green and gray choices which were used to shape the validity of all responses.

F3	**phase**	architecture	design	coding	deployment									
	action	microservices definition	api design	orchestration design	choreography design	data design	container structuring	source code authoring	network operations	code management				
	scope	uml sequence	uml component	uml deployment	uml other	data management	frontend	backend	api	messaging design	messaging deployment	container configuration	deployment scripts	github operations
	prog lang	n/a	js	js-node	python	sql	nosql db	java	yaml/json	other				
	other prog lang	<fill in>												
	aimodel	chatgpt	gemini	copilot	lmstudio-hosted									
F4	**phase**	architecture	design	coding	deployment									
	action	microservices definition	api design	orchestration design	choreography design	data design	container structuring	source code authoring	network operations	code management				
	scope	uml sequence	uml component	uml deployment	uml other	data management	frontend	backend	api	messaging design	messaging deployment	container configuration	deployment scripts	github operations
	prog lang	n/a	js	js-node	python	sql	nosql db	java	yaml/json	other				
	other prog lang	<fill in>												
	aimodel	chatgpt	gemini	copilot	lmstudio-hosted									

This constraint comes down to a limitation in how AI usage was modeled within the logging system. Each submission was designed to capture a single instance of AI use tied to one development phase, one task action, and one scope. In reality, students used an interaction with AI for multiple phases over multiple actions and tasks—for example they could ask code management assistance while working on the architectural design, or refined frontend

elements to match changes in backend endpoints. Since the submissions only allowed only one combination of phase-action-scope it made the analyze easier but it also limited the ability to reflect how the AI was used in more natural, blended or mixed ways. Additionally, some students submitted combinations that didn't formally make sense—two columns that don't align in the matrix—but in that AI interaction the student used both tasks submitted, the issue here is that the said submission would have been marked as invalid and excluded from the analysis even though they partially captured a real and reasonable usage of the developmental cycle.

4.3 Instrumentation Constraints

The data collected varied from each dataset. These changes were our own effort to improve each template per year so that the data made more sense in both the students while they were submitting their AI-interactions and to the analysis of the data. In the first course, SoftEng23, students had to choose from simple labels for their experience level or quality of AI help provided, but by the next set of students we replaced this measurement with a scale of 0 to 5 where 0 represents zero experience and bad quality and 5 master and ready to go quality.

These shifts show that the instrumentation process itself was a learning journey for our part as well. As we saw how students interacted and answered each column we adapted accordingly to new models and improved students' choices. However, this shift created some mismatches, the older labels don't line up exactly to the new numeric scales since the choices in the first course were 4 categorical but in the other courses the template was 6 choices including the zero choice. Therefore, the student's submission marked as "enough" in tool experience (with the following order "none", "some", "enough", "master") doesn't necessarily mean the value of 3.

4.4 Environmental and Contextual Constraints

All the submissions in this study were collected as part of real coursework. Students used AI tools while working on assignments, often with tight deadlines, grading pressure and changing project needs. These conditions are similar to real-world development in some ways but are still very different from open-ended projects or professional enviroments. The main goal for students was to finish their work in time, so many probably used AI in a practical, results-focused way rather than experimenting or exploring full potential.

The way the submissions were set up also may have affected the quality of what students recorded. The students had to edit a .json file, had the option to include a .txt file with the AI interaction, zip everything and upload it to the course site. Doing this repeatedly was time-consuming, and it's likely that some students skipped steps or rushed the process—not

because they weren't using AI, but because the submission system added extra work. This could explain gaps in the data or inconsistent entries that don't reflect their actual AI use.

The data doesn't show how difficult the original task was, or whether the AI's output was correct. A student might report saving time and being happy with the answer, even if it had errors. Another might werite a grate prompt, get a decent reply and feel disappointed because they expected more. Since the analysis of every prompt used couldn't happen, we cant be certain on what really happened. This means we have to take the students' own declarations at face value, treating their answers as the main source of truth—even if we can't fully verify what happened.

5. Data Specification

This chapter describes the structure, content, and interpretive logic of the dataset analyzed in this thesis. The data consists of thousands of structured interaction logs submitted by students during the execution of software engineering coursework. Each submission captures not only the technical and procedural aspects of the task, but also subjective evaluations and contextual metadata. The richness of this multidimensional schema allows for layered analysis across phases, user types, tools, and time allocated and saved.

5.1 Dimensional Fields

5.1.1 Phase for each dataset

Below is a table that shows all the choices the students could fill in 'phase' with in all datasets.

	SoftEng-23b	SaaS-24a	SoftEng-24b
phase	requirements gathering	architecture	requirements gathering
	requirements specification	design	requirements specification
	architecture	coding	architecture
	design	deployment	design
	coding		coding
	testing		testing
	deployment		deployment

5.1.2 Action for each dataset

Below is a table that shows all the choices the students could fill in 'action' with in all datasets.

	SoftEng-23b and SoftEng-24b	SaaS-24a
action	problem understanding	microservices definition
	stakeholder statement	api design
	requirements (functional)	orchestration design
	requirements (non-functional)	choreography desing
	use case specification	data design
	architectural decision	container structuring
	design decision	source code authoring
	data design	network operations
	source code authoring	code management
	unit testing	
	functional testing	

	performance testing		
	other testing		
	dev-ops		
	vm operations		
	container operations		
	network operations		
	code management		

5.1.3 Scope for each dataset

Below is a table that shows all the choices the students could fill in 'action' with in all datasets.

	SoftEng-23b	SaaS-24a	SoftEng-24b
scope	documentation (text)	UML sequence	UML state
	UML activity	UML component	UML activity
	UML sequence	UML deployment	UML sequence
	UML component	UML other	UML component
	UML deployment	data management	UML deployment
	UML class	frontend	UML other
	UML other	backend	data management
	database design	api	frontend
	frontend	messaging design	backend
	data management	messaging deployment	api
	backend	container configuration	messaging design
	api	deployment scripts	messaging deployment
	cli	github operations	container configuration
	test cases		deployment scripts
	test code driver		github operations
	test execution scripts		
	deployment scripts		
	code management actions		

5.2 Dimensional Schema

Each submission in the dataset records a single AI-assisted development interaction. The entries are encoded as JSON files, each reflecting a student's use of an AI tool to complete a specific software engineering task. The metadata fields within each JSON file span both objective and self-reported values. Objective fields include development phase, action type, task scope, tool name, programming language, and the time allocated and saved for the

task. These are selected from predefined choices, ensuring internal consistency. Self-reported fields include experience levels, perceived time savings, quality of AI assistance, emotional response, and threat perception. These are also constrained to specific scales or option sets depending on the course.

The development phase identifies where in the software lifecycle the interaction occurred, such as “requirements specification,” “architecture,” “coding,” or “deployment.” The action field specifies the nature of the work, including tasks such as “source code authoring,” “stakeholder statement,” or “unit testing.” The scope field identifies the target artifact or domain of the action, ranging from “UML class diagram” to “backend API” or “container configuration”. These three fields form a semantic triplet that defines the context of the AI usage, and they serve as the primary axes for aggregation and filtering.

Tool-related fields include the name of the AI model (e.g., ChatGPT 3.5, Copilot, Gemini), the mode of access (free, trial, full), and the user’s self-declared tool experience. Tool experience is described either with categorical descriptors (none, some, enough, master) or numeric scales (0 to 5, where 0 means no experience and 5 master experience), depending on the course. Similarly, action experience captures the user's familiarity with the type of task being performed. These two experience dimensions are essential for interpreting the data, as they influence how users perceive and rate the quality of AI support.

Each submission includes a programming language field, with common entries like Python, JavaScript, SQL, YAML, or “n/a” for non-coding tasks such as diagrams or written descriptions. This field is important because it helps show how well the AI performs in different languages. Some languages—like Python—appear more often in AI training data, so the model tends to produce better, more usable output for them. By comparing results across languages, this field helps identify where students found AI effective, depending on the language they were working with.

Time-related data in the submissions includes both actual effort and perceived benefit, as reported directly by the students. For each AI-assisted task, users entered two values in hours: how long they spent working on the task ("time allocated") and how much time they think they saved thanks to the AI ("time saved estimate"). These self-reported fields allow for a rough measure of perceived efficiency—essentially, whether students felt the AI sped things up or simply replaced work they would have done anyway. While both numbers are subjective and unverified, they offer a window into how students gauge the value of AI in practice. During preprocessing, these values were grouped and compared across users and phases to support broader analysis of usage patterns.

Subjective evaluation fields include how students rated the quality of AI help, how they felt about using the tool (e.g., “great as-is,” “needs work,” “makes no sense”), and whether they felt threatened or at risk of being replaced. These responses give insight into students’ thoughts and feelings about using AI. Quality ratings, given either as words or numbers, reflect how much they had to change the AI’s output to make it usable. Threat level, rated from 0 to 5, shows how much they worry that AI might eventually replace developer roles. The emotional response field captures their overall attitude—whether they see the tool as helpful, promising, or flawed.

5.3 Measurement and Normalization

Each submission combines fixed-choice selections with self-reported reflections, making it possible to compute a wide range of metrics during preprocessing. Aggregates like average time allocated, average time saved, and dominant experience levels overall and per phase are calculated per user. These allow comparisons across groups—such as inexperienced versus experienced users—or over time, especially for students who participated in more than one course.

Alongside usage metrics, each student is tagged with additional demographic metadata derived from their institutional username. This includes an estimate of their year of entry and corresponding age group. For instance, a username like "el21xxx" implies entry in 2021, from which approximate age can be inferred by adding 18 to the gap between the current and entry year. These derived values support group-based analysis by AI exposure—whether a student started before, during, or after the mainstream availability of AI tools, thus creating the category AI exposure and labeled as preAI, someAI or AIsince start. The students marked as preAI were the ones who started studying from 2019 and before, someAI were the students who started between the years 2020 and 2022, whereas the marking AIsincestart was for the students who started after 2022. While these fields are proxies and not self-declared, they offer useful context for tracking patterns in behavior and perception across different academic generations.

5.4 Objective vs Self-Reported Fields

A critical distinction in the dataset is between fields that are objectively structured and those that reflect subjective assessment. Objective fields include phase, action, scope, tool, language, and time spent. These are constrained by input design and validated through the filtering pipeline. As such, they form the structure of the dataset’s analytic stability. In contrast, self-reported fields—experience levels, quality, emotional feeling, threat level, and time saved—are declared by the student’s estimate. They are influenced by user expectations, task success, project stress, and familiarity with AI interaction paradigms.

Fields like age, AI exposure and the year the student started studying are derived from structured data and treated as objective. However, self-reported fields—such as experience level, perceived quality, emotional response, and time saved—are clearly subjective. While they are subjective, these fields are treated as accurate representations for analyzing.

5.5 AI Tool Use and Variations

The dataset includes a range of AI tools used by students, but in reality, most submissions are concentrated around a few key models. ChatGPT dominates across all three courses, with both version 3.x and 4.x appearing frequently—by far the most used tool, regardless of task type or development phase. Its widespread use likely reflects a combination of accessibility, familiarity, and general-purpose capability. Other tools like GitHub-copilot, Gemini, Bard IntelliJ integrations or LM studio-hosted models appear in a lower frequency.

Each submission records not just the AI tool's name but also its access level—whether the student used a free, trial, or full version. This matters, especially for ChatGPT, where the version used (e.g., 3.5 vs. 4.0) can lead to noticeable differences in response accuracy and quality. The dominance of ChatGPT also creates a statistical issue—tools outside this cluster appear so little that it's difficult to make strong comparisons. In most cases, there just isn't enough usage to generalize about effectiveness, preferences, or reliability for non-ChatGPT tools.

The reasons why students used more ChatGPT are not explicitly recorded, but a few patterns are likely. Access constraints may have played a role—some tools are gated behind paywalls. Peer influence recommendations may have also shaped the patterns for the use of ChatGPT, especially in fast-paced group settings where one student's tool becomes the default for others. Even when multiple tools were available, most students seemed to prefer sticking with what they knew.

Despite this imbalance, the combination of tool metadata with the task type, programming language, and self-reported outcomes through analysis captures the same goals that we aim to capture. It supports the targeted analysis on how the tool choices interact with student's prompts, perception and perceived quality outcome. Since ChatGPT has the most submissions by a range from 85-95% for each database set, it defines the reality of what this thesis actually analyzes—since it's by far the most used tool across all submissions.

6. Sample Identity

The empirical strength of this thesis rests on the scale, consistency, and segmentation of the student population from which its data is drawn. This chapter describes the properties of that population in full: the number and type of participants, the volume of submissions they generated, the derivable demographic indicators encoded in their identifiers, the classification of their experience levels, and the longitudinal structure that allows intra-subject tracking across time. The goal is not merely to count users but to expose the analytical logic that transforms raw participant data into structured identity groupings, each of which supports specific claims about how AI tools are used and perceived in educational software development.

6.1 Population and Submission Characteristics

The population size is internally varied. In softeng23b, 126 students participated. In saas24a, participation dropped to 79, and in softeng24b, participation reached its peak, with 176 students contributing submissions. These raw participant counts are the total number of submitted interactions. Below is a table that shows each number for each dataset.

	SoftEng-23b	SaaS-24a	SoftEng-24b	Total
Number of students	126	79	176	330
Number of submissions	1,540	1,184	3,175	5,899
Valid submissions	1,029	763	2,060	3,852
Gray submissions	215	163	634	1,012
Invalid submissions	296	258	481	1,035

It is important to note that these numbers represent submission counts, not users. Each student could submit multiple entries depending on the complexity of their project and the phases they chose to document. The granularity of the data is therefore intra-individual. This means that user-level inferences are built not from single-point responses but from patterns distributed across multiple structured records per student. This allows for much more stable statistical aggregation and the detection of longitudinal or behavioral consistency.

6.2 Deriving Demographic Attributes from Identifiers

Each student's NTUA institutional username contains an embedded numerical indicator of the year they began their studies. The format of these usernames follows a standardized pattern—a prefix of "el" followed by two digits indicating entry year, and then the personal ID for the student. For example, "el20xyz" would denote a student who started their studies in 2020. Using this embedded information, it is possible to derive the three demographic proxies, year of entry, estimated age and AI exposure.

The year of entry is a stable identifier that allows grouping students by educational generation. This grouping enables comparative analysis, particularly when examining the influence of AI availability on student behavior. For age estimation, a fixed academic entry age of 18 is assumed. From that baseline, estimated age is calculated by subtracting the entry year from the year the data was collected—so 2023 in this case—and adding 18. While this method does not capture non-traditional entry paths or academic delays, it provides useful age band estimations (e.g., 20–21, 21–22, 23–24) for demographic visualization. Also, the level of AI exposure which categorizes the user's exposure to AI models based on when the most AI tools became easily accessible and mainstream.

This form of demographic inference requires no self-declaration from each student, and maintains pseudonymity. It enables analysis of whether younger students with some or complete exposure to AI models display different usage behaviors or perceptions compared to older students who didn't have AI tools since their enrollment.

6.3 Encoding and Aggregating Experience Levels

Experience is one of the primary variables used to divide the dataset and is captured along two main axes: task experience and tool experience. Task experience, or action experience, shows how familiar the student is with the type of development activity logged (e.g., unit testing, architecture modeling). Tool experience reflects how comfortable or skilled the student is with the specific AI tool used during the task—that could mean better prompts given or expected when that AI tool is no help in some tasks.

In softeng23b, these values are recorded as textual categories—"none," "little," "fair," "big" or "master". In saas24a and softeng24b, the experience fields are encoded numerically on a scale from 0 to 5 where 0 represents no experience and 5 master experience.

Experience encoding is not evaluated at the level of single submissions. Instead, a student's dominant experience level is calculated by aggregating across all their entries. The most frequent response value per user, for both task and tool experience, is retained. This aggregated label reflects the user's self-perception. The two labels—tool experience and

action experience—are then concatenated into a single column-identifier. Students were labeled as experienced if they showed a level of confidence in experience for both the AI tool they used and the type of task they were working on. For the courses that used numerical scales (like saas24a and softeng24b), this meant scoring 3 or above out of 5 in both tool experience and task experience. For softeng23b, which used categorical labels instead of numbers, students needed to report at least "some" or "enough" experience in both fields to be considered experienced. Anyone who fell below these thresholds in either category was labeled as inexperienced. This classification helped distinguish between users who were comfortable with both the tool and the task, and those who were still building familiarity in one or both areas.

This binary segmentation allows the analysis to track whether perceptions of AI quality, threat, or time savings differ between users with deep experience and those still exploring their capabilities. It also helps analyze the exposure of the students who appear in both SoftEng-23b and the second course SaaS-24a, revealing how their initial categorization as experienced or inexperienced in SoftEng-23b aligns with their responses in the dataset of SaaS-24a.

6.4 Constructing AI Exposure Categorization

In addition to task familiarity and tool confidence, students are classified according to their exposure to AI tools across their educational timeline. The logic here is temporal rather than skill-based. Since large language models such as ChatGPT became widely accessible in late 2022, students who began their studies before that date would have completed multiple semesters without AI tools. In contrast, students who entered in or after 2022 would have had continuous exposure to such tools from their first year.

Based on this reasoning, students are grouped into three AI exposure categories. Those who began their studies in 2019 or earlier are labeled as PreAI, meaning they had no AI exposure during their early academic training. Students who started in 2020 or 2021 are labeled as SomeAI, representing partial overlap between traditional and AI-integrated software engineering education. Students who entered in 2022 or later are classified as AIsincestart. These students represent a new generation for whom AI tools are a built-in component of the development environment rather than an external enhancement.

This classification allows the analysis to test whether early or continuous exposure to AI shifts patterns of tool selection, quality evaluation, or emotional stance. It also supports curriculum-level reflections: if AIsincestart students display huge differences in interactions with AI tools, this may signal a grave error in how teaching happens currently.

6.5 Students Who Enrolled in Both Courses

The most analytically powerful segment of the dataset consists of students who participated in both softeng23b and saas24a. These are 51 students who engaged with the same AI logging system across two distinct academic periods. For each of these students, it is possible to track changes in declared experience, emotional sentiment, perceived quality, and threat evaluation across time.

For each of these students, their experience declarations in SoftEng-23b are the reference used while analyzing their answers in SaaS-24a. Each student was marked as experienced or inexperienced and then was compared to the data they submitted in SaaS-24a, to compare their experience level, declared quality of AI help, perception of threat level, and their generic feeling for the use of AI.

The main goal was to find any patterns between the declaration of the inexperienced students that contrasted with those of the experienced users. For example we analyze the perceived knowledge acquired and quality of AI help by both the experienced and inexperienced users to ascertain if the inexperienced users find AI use better or if they learn more than the experienced users.

Since SoftEng-23b is mandatory for the students who select flow-L in the school of Electrical and Computer Engineering and SaaS-24a is not we expect to find more students in the first course who mark themselves as inexperienced, whereas the students who enrolled in both courses likely had grater prior interest in software development, thus more categorized as experienced.

7. Methodology

This chapter provides a detailed, step-by-step account of the processes and systems used to turn raw student submissions into structured, analyzable data. The goal was to transform decentralized, user-submitted logs into a dataset that supports clear, phase-specific, and user-level insights. To achieve this, the methodology followed a strict pipeline: beginning with submission, continuing through validation and enrichment, and ending in the export of visualized findings. Every phase of this transformation was grounded in reproducible scripts and structured schemas. The entire pipeline was built to support tracking, slicing, and comparisons across technical, emotional, and demographic axes—all while preserving the original per-user granularity of the submissions.

7.1 Submission Infrastructure and Data Acquisition Context

All data was collected as part of the formal course requirements for two advanced software engineering classes over three semesters at the National Technical University of Athens. In each course, students were required to submit AI-interaction logs that documented their use of AI tools during the development cycle. They were captured at the moment of interaction, concurrent with the phase they were being executed in. This ensures that the data reflects actual usage, not reconstructed memory or post interpretation.

Students submitted their work through a standardized web-based interface designed specifically for this purpose. The interface accepted only zipped archives containing a combination of structured metadata as a .json file and optional prompt logs in .txt file. Each archive was required to contain one valid .json file, which followed a pre-specified schema tailored to the course in question. Optional .txt files could be added to document the exact prompt or AI response and conversation used in the task, but these were not mandatory and were excluded from the quantitative analysis pipeline. However they were used in order to specify the correct validity path in each phase, that was used in the filtering section. The interface included basic server-side validation to ensure that the zip file submitted was not corrupt and that the .json file could be parsed and included all columns that were asked. Submissions without a readable .json file were rejected automatically at upload time, forcing the student to resubmit.

Each student had to login with their institutional username and password in order to submit each AI-log, and were limited to 10 total logs per day to ensure that they wouldn't submit duplicate entries. Once uploaded, each archive was unpacked on the server side and stored in database for each course-set. This ensured traceability across sessions and allowed mapping between user metadata (extracted from the username) and course submission contents. Timestamps were preserved from the upload metadata to allow future

chronological ordering of submissions per user if needed, although this temporal ordering was not used in the current study. Every valid .json file was then passed into the cleaning and validation pipeline in Tableau Prep.

7.2 Validation Pipeline and Matrix

The transformation of raw submission data into analyzable metrics required more than simple cleaning. A sequence of structured steps was applied to each entry, producing new fields that formed the foundation of cross-phase, cross-experience, and per-user analysis. These operations are categorized into five distinct and important stages: mapping of experience, combinations per phase, aggregation of time values at the user level, determination of dominant experience labels and extraction of demographic metadata from usernames.

Each of these steps was implemented using Python, through the library of pandas - DataFrames, and executed via Tableau Prep for schema alignment and traceability. The resulting fields were then injected back into the unified dataset for further filtering, slicing, and diagram export.

7.2.1 Phase–Experience Combinations

The first layer of transformation involved resolving each student's experience level with specific tasks and tools, segmented by development phase. Because students could submit multiple entries per phase—each involving different tools or actions—it was necessary to identify their prevailing experience level for each combination.

To accomplish this, the dataset was grouped by username and phase. Within each group, all reported action experience values were collected and evaluated using a predefined priority order. This priority order established that "none" ranks lower than "little," which ranks below "fair," and "big" ranks highest for the database of SoftEng-23b, in the other databases the ordering was from 0 to 5. Ties were resolved by selecting the higher-priority category, ensuring that stronger declarations were the assigned label. The same logic was applied independently for tool experience.

The result was two new fields: one recording the most frequent action experience per user per phase, and one recording the most frequent tool experience per user per phase. These fields allowed later visualizations to display experience-modulated perceptions in phase-specific contexts, such as how threat level varies between inexperienced and experienced users in the entire software development cycle.

7.2.2 User-Level Time Averages

Time metrics—specifically time allocated and time saved—were originally recorded per submission. These values, while useful at the local level, needed to be aggregated upward to support user-level and group-level analysis. Two average values were computed for each user: the mean time spent across all their submissions, and the mean time they reported saving as a result of AI assistance.

To avoid distortion by outliers or uneven submission counts, only validated entries (valid or gray) were included in these calculations. These averages allowed the construction of visualizations comparing the basis of reported efficiency, including diagrams that show whether experienced users tend to save more time or whether certain phases consistently result in underestimation or overestimation of AI benefit.

The output of this step consisted of two new fields: users_average_time_allocate and users_average_time_saved. These fields were joined back to the per-submission records so that phase-level and group-level comparisons could be rendered in the same visualization layer.

7.2.3 Dominant Experience Labels per User

To understand the experience level of each student, the categorization was based on their experience level in total—not just per phase but for all their submissions. This allowed to divide the students into the two categories of experienced and inexperienced. Thus, analyzing and comparing perceived AI quality, threat level and generic feeling.

All reported experience values for each student were collected for both the tool use and action familiarity, the most frequently declared value, either categorical or number based on the dataset was kept for that student. In a case of tie the highest value was kept to avoid underrepresenting the student's experience.

Based on this, two summary labels were created: tool_experience and action_experience indicating the overall experience for the student on each category. This method made it possible to distinguish students working mostly for the first time in either the task used or with the AI tool used.

7.2.4 Composite Experience Category (TE_AE)

The most useful label for analysis was a combined one that joined tool and action experience into a single tag. This was called **TE_AE** and formatted as "TE: {tool_experience} AE: {action_experience}".

This label was used to sort users based on experience. If both experience scores were 3 or higher the user was marked as experienced and in the first database—experienced users were the ones who submitted in both categories values higher than “some” and “enough”. If either score was lower, the user was considered as inexperienced.

This categorization was used in all comparisons between experience groups—such as analyzing differences in quality ratings, perceived threats, or knowledge acquired. It also made it possible to track how students moved between categories over time as they became more confident and skilled from the dataset of SoftEng-23b to that of SaaS-24a.

7.2.5 Demographic Derivations from Institutional Usernames

Since the demographic fields were inferred indirectly, through the student’s username, the students didn’t have to provide additional personal data. Each student’s institutional username followed the format for ECE – NTUA’s school (e.g., el21XXX), where the embedded number indicated the year they started studying. By decoding the enrollment year three main new columns, the estimate age, starting year of studying and AI exposure.

The age estimate was calculated by assuming that all students enroll at the age of 18. Then the age derivation was a subtraction of the year the dataset was collected—2023 with the year the student started studying and the addition of 18. Also, based on the enrollment year the AI exposure was marked. Students who enrolled before 2019 were labeled as preAI, 2020-2021 as someAI and all students after 2022 as AIsincestart.

All such derived values were used for grouping, filtering, and visualizing across student responses—enabling comparisons based not only in responses but also in a demographic context

7.3 Derived Metrics and Aggregation Logic

Once the cleaned dataset was established, several new fields were derived per user and per submission. These derivations enabled user-level aggregation and comparison across experience, phase, and sentiment categories. All derivations were implemented in Python using Tableau Prep to ensure the creation of the new fields through the database and to visually verify the correctness of the applied methods.

The first and most important process was the identification of each student’s experience level. Instead of focusing on a per phase basis or per scope, the entirety of submissions was taken into consideration for each student—based on the most frequent answer given by the user. This was filtered at the threshold of both action and tool experience being above 3, all experienced students were filtered through the TE_AE column in Tableau Desktop (e.g., “TE: 4, AE: 3”. After filtering through Tableau Desktop, the diagrams were created displaying the

quality of AI reported, generic feeling, threat level, and languages used for both the experienced and inexperienced students.

A second set of derivations focused on time. Although every submission included a student's estimate of time spent and time saved, these new values were the averages per user. Therefore in each submission there are two new columns for the user who submitted the log, the average time saved and average time allocated as a constant throughout all the submissions made by that student for better analysis.

Demographic context was also inferred based on the student's username, thus estimating the age group, level of AI exposure also referred to as "generation" and the year they started studying. This allowed the data to be grouped along generational lines—with the most important division being based on how much exposure they had for the use of AI tools.

All these new columns of data was added back to the dataset and became the basis for filtering, visualizing diagrams and comparing them throughout the results. They enabled reliable filtering, and tracking.

7.4 Visualization Process and Diagram Export

All visualizations presented in the thesis were generated using Tableau Desktop, following a strict schema alignment between the master dataset and the diagram-specific extracts. Each diagram corresponds to a filtered subset of the full dataset for each of the three courses that provided the submissions, focused on a particular metric, group, or comparison. For example, diagrams related to perceived AI quality in pre-development—pre-code related phases only include submissions tagged with those phases and exclude all others. Filtering, grouping, and normalization were applied directly in Tableau to preserve visual traceability.

In Tableau Desktop, four new calculated fields were created in each workbook to standardize how data is visualized across all diagrams. These fields—named "percentage", "percentage per username", "count per username" and "count—serve different purposes depending on what each chart aims to display. "Percentage" displays the percentage per all the submissions based on the submissions in the given database calculated by the count shown over the count of the total submissions with two decimal places. "Count per Username" tracks how many unique students fall into each category based on the filters applied in a given diagram. Building on this, "percentage per username" is the label of each category as a proportion of the total user count, thus, showing the percentage on the diagram for better visualization.

Each diagram was then exported as a png and a csv containing all the necessary columns so that the diagram could be shown in the frontend webpage provided for the readers. These extracts were named using a strict naming convention that aligns with the diagram IDs used throughout the thesis (e.g., D42, D67). These CSV include only the data to display the diagram visually, not each students submission; those datasets are included in the derived csv files from Tableau Prep and are also open sourced to any reader for transparency and use.

7.5 Deployment of Diagram Delivery Platform

To make the diagrams accessible to the reader, a custom frontend web page was built hosting all diagrams created by this project. By using Highcharts, this interface was designed to replace the static exports of Tableau Desktop for better exploration of the thesis findings. Users can switch between course datasets—SoftEng-23b, SaaS-24a, SoftEng-24b and the intersection of users in both SoftEng-23b, SaaS-24a. This allows students, readers, and researchers to analyze the results, and validate insights without having the need to use Tableau environment.

The entire structure of the frontend is organized into the four main databases, and in each database there are six categories, each corresponding to a specific part of the thesis, Overview, AI Usage Context, Pre-Coding Related Phases Analytics, Pre-Coding Related Phases Experience Perspectives, Coding Related Phases Analytics, Coding Related Phases Experience Perspectives. Within each of these categories, subcategories are displayed, each with a short explanation and a label indicating the number of diagrams it contains. For example, under “Overview”, users will find subcategories like “Participant Demographics” with six diagrams some of them showing age, entry year, and validity distributions.

To support this structure, each diagram’s data was first exported through Tableau Desktop’s interface as both a csv file and a png. Each csv file contained all the information needed to display the diagram through highcharts—columns include x-axis, y-axis and labels in each distribution set. With the use of a custom Python script, a json map was created to support the schema expected by highcharts for each diagram. As a result, each diagram displayed on the site is directly linked to the data exported from Tableau Desktop and is rendered dynamically from its corresponding json map.

The result is a web platform where users can view the diagrams, download them, and interact with each filter. The use of the frontend could be instructional feedback, comparison, or research reuse, it offers a complete and transparent interface that displays all the diagrams that were used in this thesis’s analysis. Every design choice for the frontend

display—from category layout to interactive filtering—was made to reinforce the adaptability and exploration of the webpage as easy as possible to a user-friendly level.

8. Results

This section presents the results of the study, structured to answer the research questions defined in Section 2.3. We divide the analysis into five parts: data integrity (perception and noise), AI use in code-related phases, AI use in non-code-related phases, longitudinal patterns among tracked students, and the influence of programming language and experience. For each section, we list and interpret selected diagrams and explain how the results respond to the research questions.

8.1 Perception and Noise

Before analyzing patterns of AI use, student reactions, or how effective AI was in different phases, it's important to make sure the data is consistent, clear, and trustworthy. During preprocessing, each submission was labeled as valid, gray, or invalid—these categories reflect how clearly and carefully students documented their use of AI. Additionally, since AI logging was tied to the grading system and was not optional, many students submitted entries without filtering what they submitted, thus many multiple entries were marked as invalid due to duplicated entries. The results here respond to Research Question RQ0: Can student-submitted data be reliably used for analysis?

Figure 1.7.1 presents the distribution of validity states across all phases for the SoftEng-23b course. Coding emerges as the most structurally robust phase, with a clear majority of submissions falling into the valid category. This isn't surprising since coding tasks are usually clear-cut, with defined inputs and outputs, and the kinds of help AI provides—like generating functions, entire scripting files or solving syntax issues—are easy to describe in the schema. In the requirements phase, a higher percentage was marked as invalid because students struggled to map abstract tasks like stakeholder analysis or functional or non functional requirements. They often mislabeled actions or scopes, showing confusion about how to log AI use in these phases.

Figure 2.7.1 displays validity distribution for the SaaS-24a dataset. Here, the pattern holds for coding, but the overall validity drops in phases such as architecture and design. These phases exhibit a substantial rise in invalid and gray-tagged entries; The architecture phase in particular, has only around 60% of submissions marked as valid. Same patterns appear as the first dataset, gray-marked submissions, seems to highlight a mismatch between how students think about their work and how strongly the defined schema markes valid inputs, rather than indicating it as misuse, the high number of gray entries (~15% for each phase) point to structural ambiguity for all the submissions in the databases.

In Figure 3.7.1, presents the validity distribution for the SoftEng-24b dataset. The same pattern appears, coding remains the most stable phase with the highest validity percentage submission. While the requirements gathering and specification show the highest concentration of invalid percentage per phase, from requirements gatherings having 65%

valid responses and requirements specification at 55%. Like previous datasets these low percentages suggest confusion around how to log AI use in early, non-code related phases.

In all three course datasets, deployment stands out for having the least number of valid submissions. Even when students document AI use in this phase—such as for container scripts or environment setup—the entries are often marked invalid or fall into the gray category. A common issue is mislabeling: for instance, many students classify frontend work as part of the design phase, when in fact, design refers to the planning and structuring of system components—like defining data flow, component interactions, or selecting architectural patterns—not implementation-level details like UI styling or layout coding. Unlike coding, where AI produces clear, testable output, or testing, where results are straightforward to evaluate, deployment involves less visible tasks—like editing configuration files, or resolving dependency issues. These are harder to classify within the current phase–action–scope model and may require future adjustments to better capture the nature of operational tasks.

Another issue was regarding the initial—pre-code related phases where the students' responses were usually marked as gray or invalid, and had very little response number of submissions in those phases. This pattern suggests more than just the misclassification; it may show an uncertainty about the logging of AI use during the initial phases. Students may not have submitted their initial AI interactions such as brainstorming, drafting initial ideas, definition of requirements—by thinking they were not as worth to document. Another trend across all courses is that the number of gray entries in design and architecture phases gets smaller from SoftEng-23b to SaaS-24a. This likely shows that students are getting better at categorizing their AI interaction with the corresponding phase-action-scope. With repeated exposure, they seem to learn how to classify their AI use more accurately.

In summary, after the filtering is applied and we exclude the invalid responses, we can more safely conclude that the submissions left are structurally reliable and have true meaning since they are aligned and checked with the validity path created. As a result, the filtered dataset offers a clear view on how students used AI across different development stages, minimizing noise from confusion or rushed submissions.

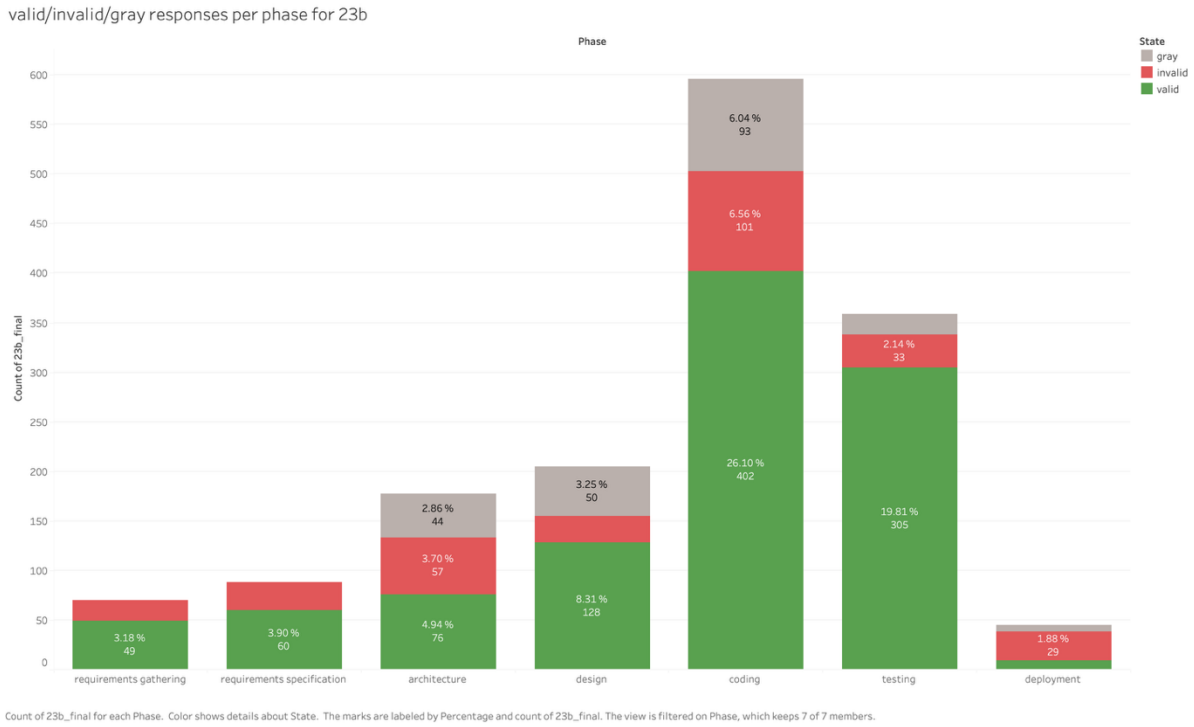

Figure 1.7.1 Validity distribution per phase in softeng23b

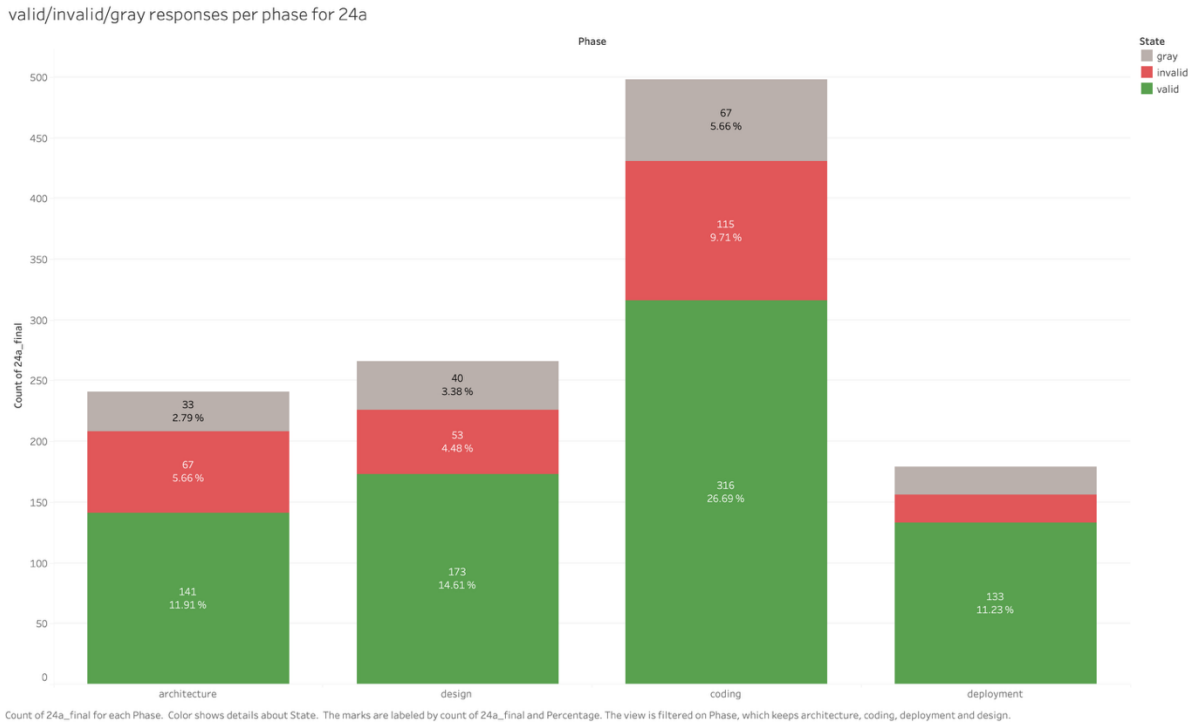

Figure 2.7.1 Validity distribution per phase in saas24a

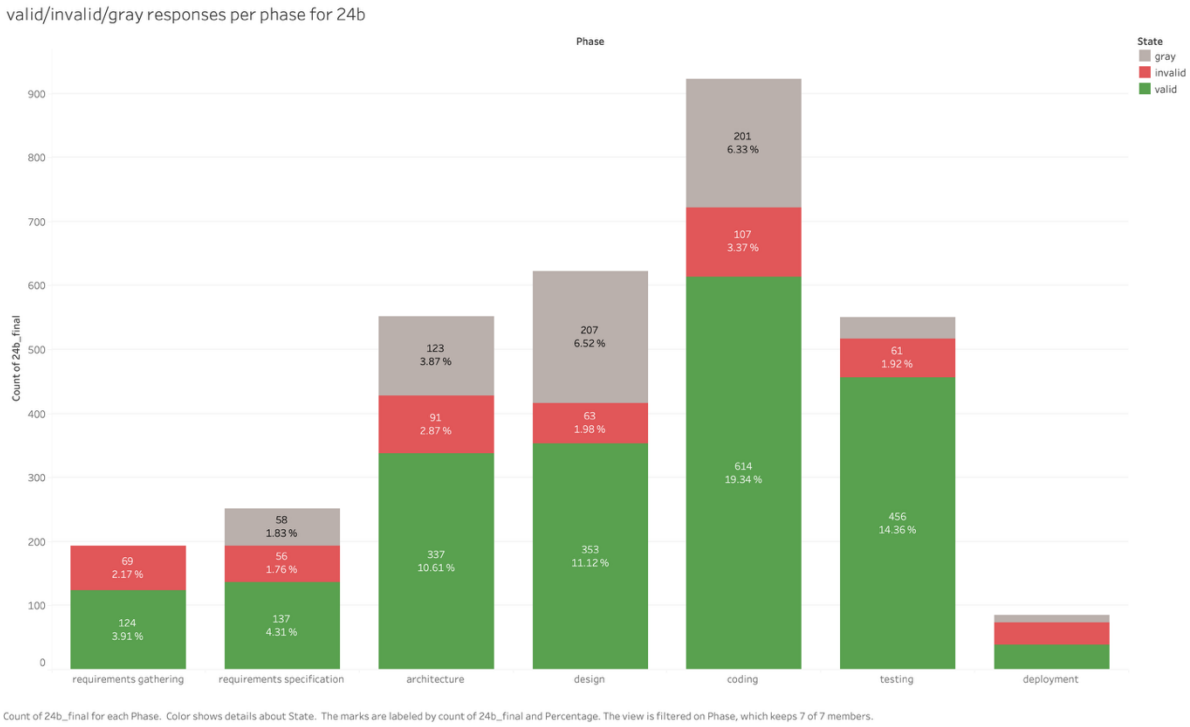

Figure 3.7.1 Validity distribution per phase in softeng24b

8.2 Code-Related Phases

This section investigates AI tool usage during code related stages of the software development process: coding, testing, and deployment. These phases collected the most amount of submissions since the students used AI tools mainly in the phases of coding and testing—where source code authoring, and code management was major. This section mainly focuses on the RQ2 for the analysis of AI use during the code related phases—but also aids the analysis of RQ4.

The coding phase dominates the AI interaction landscape across all three datasets. In Figures 4.7.2, 5.7.2, and 6.7.2, which display declared quality of AI help during code-related phases for SoftEng-23b, SaaS-24a, and SoftEng-24b respectively, student responses consistently cluster at the top of the quality scale. Most entries in the coding category are rated as either "ready-to-use" or requiring only "minor modifications", or have high reporting at 4-5 from a scale of 0 to 5. This signals a deep level of match between what the student expects and AI output quality in this phase. It's clear that students have learned how to prompt for coding tasks in ways that get solid results. Since the LLM's have a huge training set in programming languages like Python or JavaScript, the output given at any coding help provided will be ready-to-use for both the creation or the syntax fix on a file.

In contrast with the coding phase, the testing phase reports a slightly less quality in AI output. For the dataset of SoftEng-23b, testing marked as second highest in submissions, and the distribution of quality of AI help in that phase was slightly skewed to the right—showing a difference between the coding phase. Testing was registered as "minor modifications needed" for the most part. While in SoftEng-24b we see notice more submissions around the values of 3-4 indicating a moderate quality in those phases.

Deployment has the least amount of submissions in the SoftEng datasets, but even in those submissions we notice that the most answers categorize as the quality of AI help as "minor modifications needed". Whereas in the SaaS-24a dataset we notice the same pattern of a moderate rating of AI quality and the most submissions being at the values of 3-4.

Students' reported time savings follow the same pattern as their quality ratings. Figures 7.7.2, 8.7.2, and 9.7.2 show how much time they said they saved compared to how much time they spent on tasks like coding, testing, and deployment. It's clear that for all datasets and all phases the AI was a help, since the time saved is always nearly double than the time allocated. The most extraordinary finding is that the average time allocated in the testing phase in the dataset of SoftEng-24b was around 0.68 hours and the average time saved is close to 1.54 hours which is way more than double the time allocated. This could mean that the students reported higher time savings in the testing phase not just because the AI helped provide code, but because testing often requires the understanding of defining edge cases, understanding how components should interact and so on. Therefore, AI helped clarify more than just code—which students already comfortable writing—and since in the coding phase

only code was provided as outcome, then, students expectedly reported more time savings in testing phase.

When it comes to how students feel about AI as a possible threat, the response depends on the task. Figures 10.7.2 and 11.7.2 show how experienced users in SaaS-24a and SoftEng-24b rated their level of concern for replacement. Most of them sit in the range (1–2), which shows they're not cautious nor alarmed. They seem to accept that AI can take over simple or repetitive code tasks, but they don't think it's about to replace them entirely. Their hesitation is more about responsibility: they're fine letting AI handle structure or syntax, but they're not ready to hand over important design or performance decisions to something they can't fully understand or control.

In contrast, Figures 12.7.2 and 13.7.2 capture the declared threat levels of inexperienced students in the same phases. These distributions are heavily skewed toward the low end of the scale (0–2), suggesting that students perceive their AI interactions as far from a threat. For these users, AI is not a potential replacement but a helping tool—helping them fill knowledge gaps, accelerate execution, and build confidence in tasks that might otherwise be inaccessible or intimidating. Their threat perception is low because AI support is framed not as competition but as a temporarily provided aid—which seems wrong because these phases are under the highest threat of AI replacement.

The difference in how experienced and inexperienced users react emotionally points to the understanding of phase or action, the more they realize what they're handing over to the AI—they judge more carefully whether that handoff actually makes sense. Inexperienced users tend to see AI as a helpful boost, where experienced users see it more as a tradeoff—useful, but with limits and risks that need to be managed.

Collectively, the code-related phases formed the core of AI interaction across all datasets, both in number of submissions and perceived quality. Coding was consistently rated highest in quality with "ready-to-use" code provided. While testing, rated slightly lower than coding in quality, it actually showed a higher time saved-time allocated ratio, likely because AI besides providing code for that action, it also provided understanding on why the structuring task had to be altered in a specific way. Deployment reports limited use, the quality provided was moderate. Across all phases the time saved-time allocated ration was always close to double. Experienced users declared a low threat of AI, whereas the inexperienced users declared strongly the lowest threat—suggesting their inability to see the threat that AI poses in these phases.

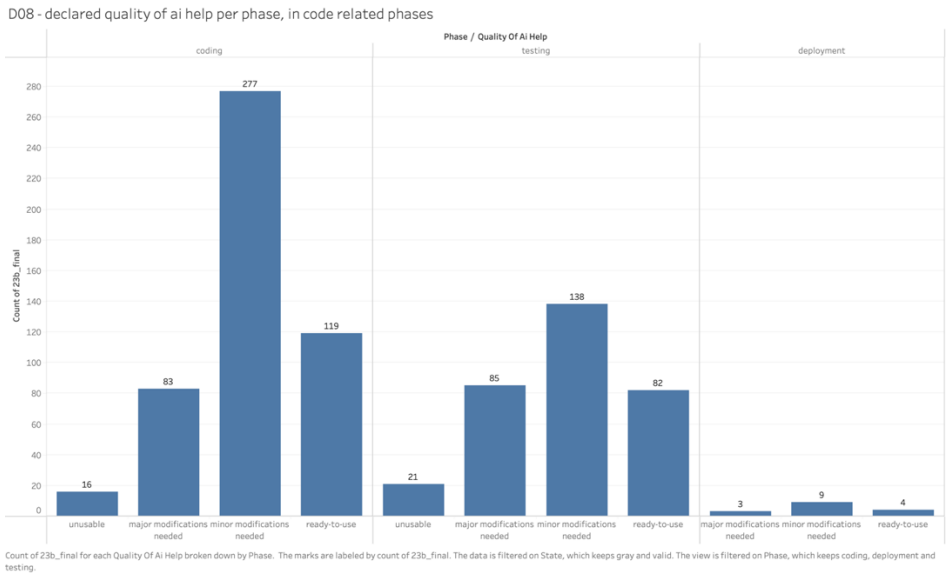

Figure 4.7.2 Declared quality of AI help per phase in code-related phases (softeng23b)

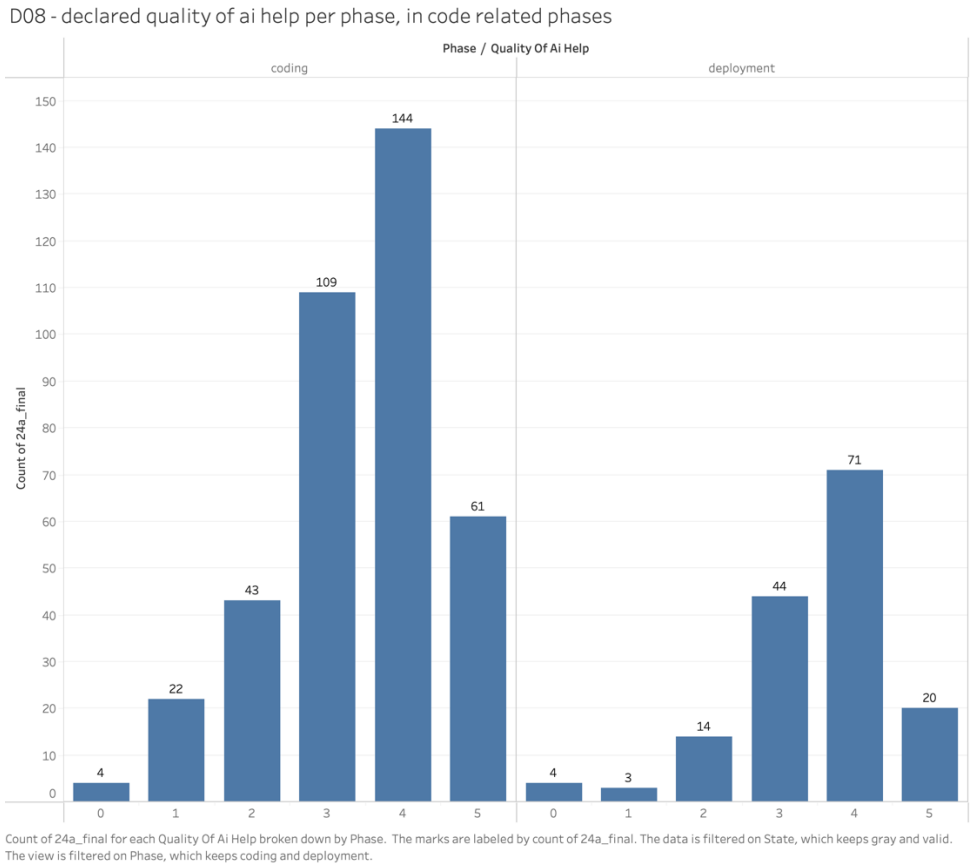

Figure 5.7.2 Declared quality of AI help per phase in code-related phases (saas24a)

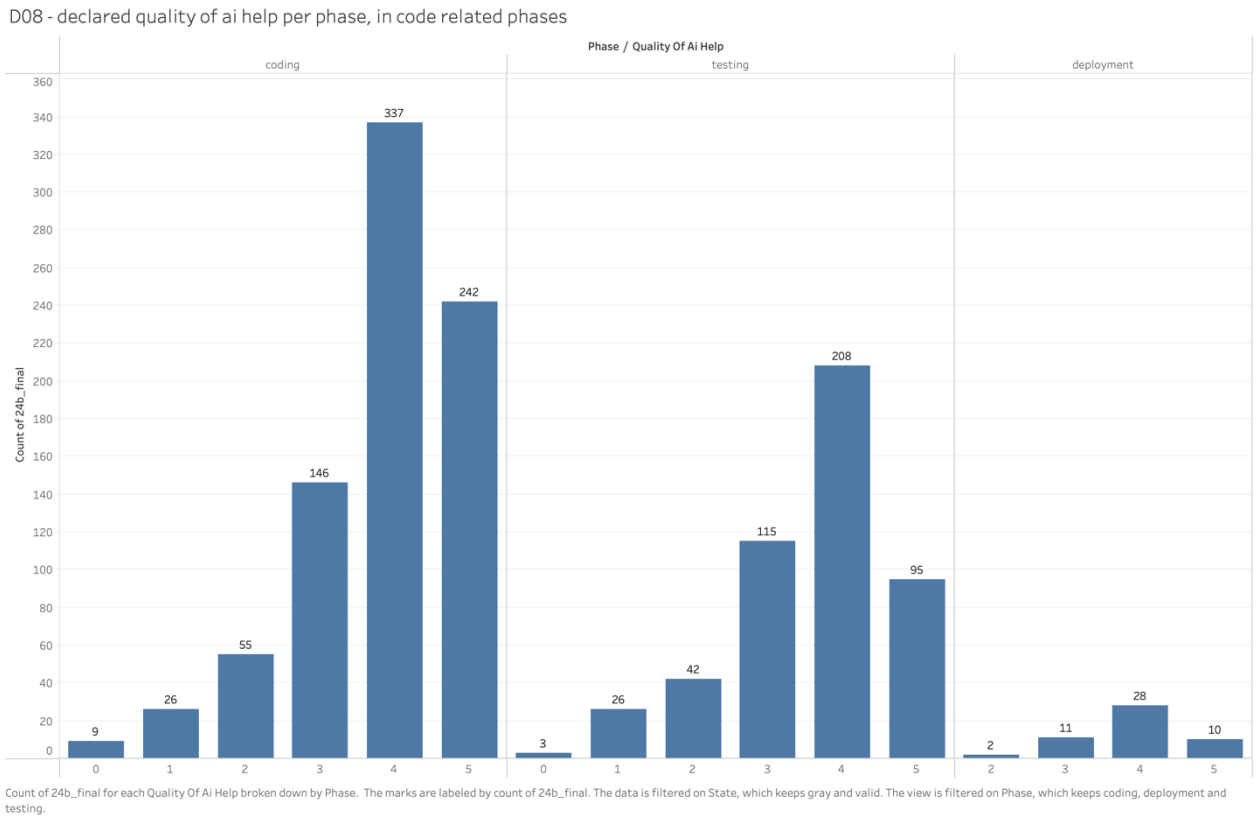

Figure 6.7.2 Declared quality of AI help per phase in code-related phases (softeng24b)

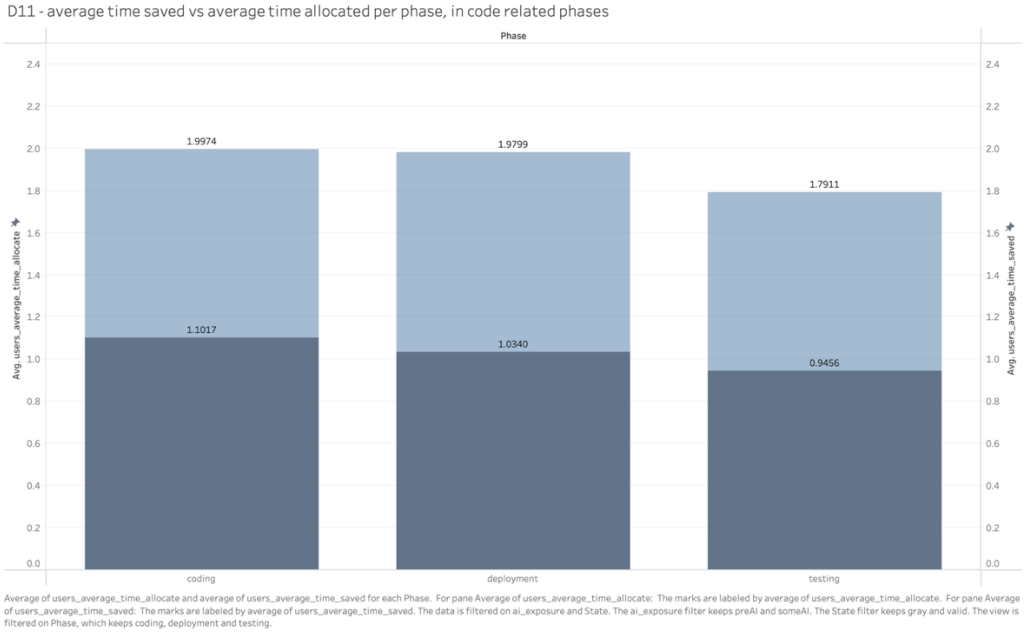

Figure 7.7.2 Average time saved vs average time allocated per phase in code-related phases (softeng23b)

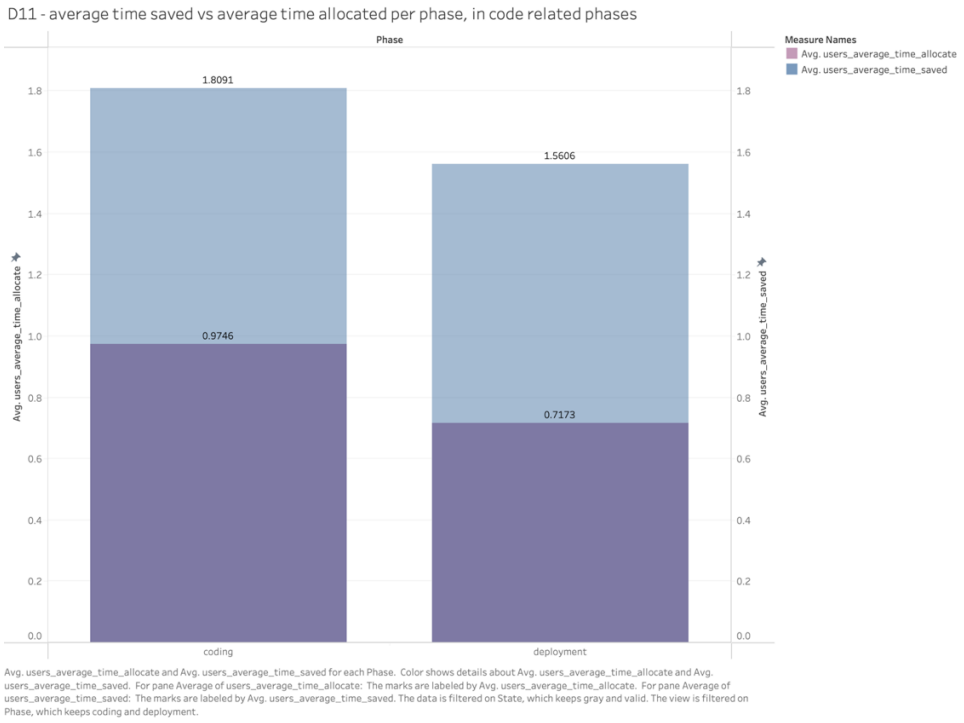

Figure 8.7.2 Average time saved vs average time allocated per phase in code-related phases (saas24a)

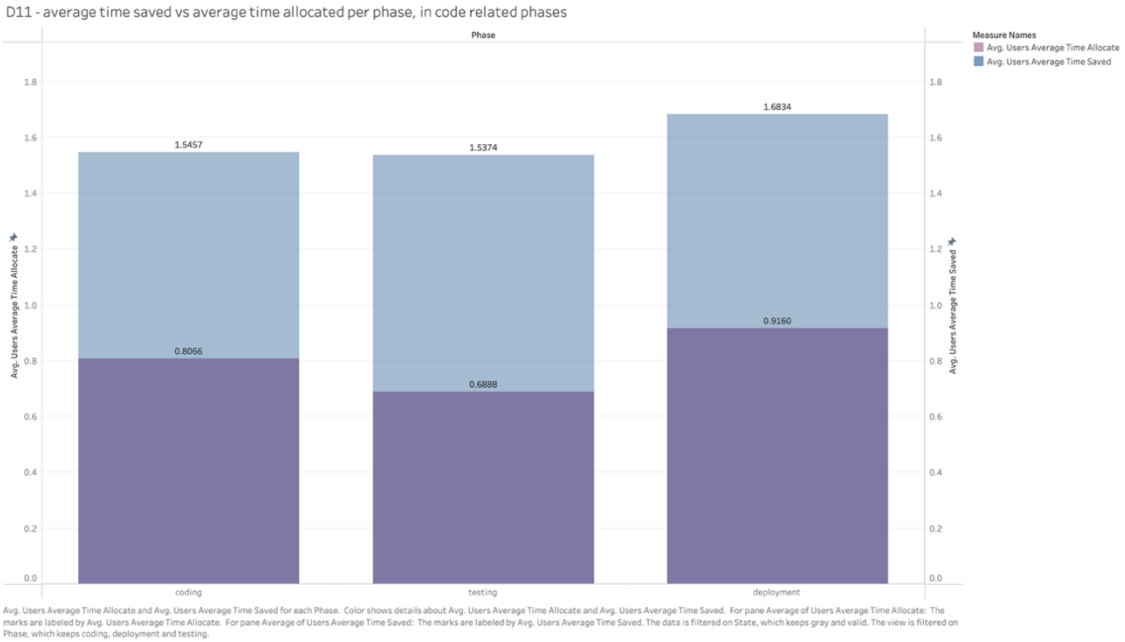

Figure 9.7.2 Average time saved vs average time allocated per phase in code-related phases (softeng24b)

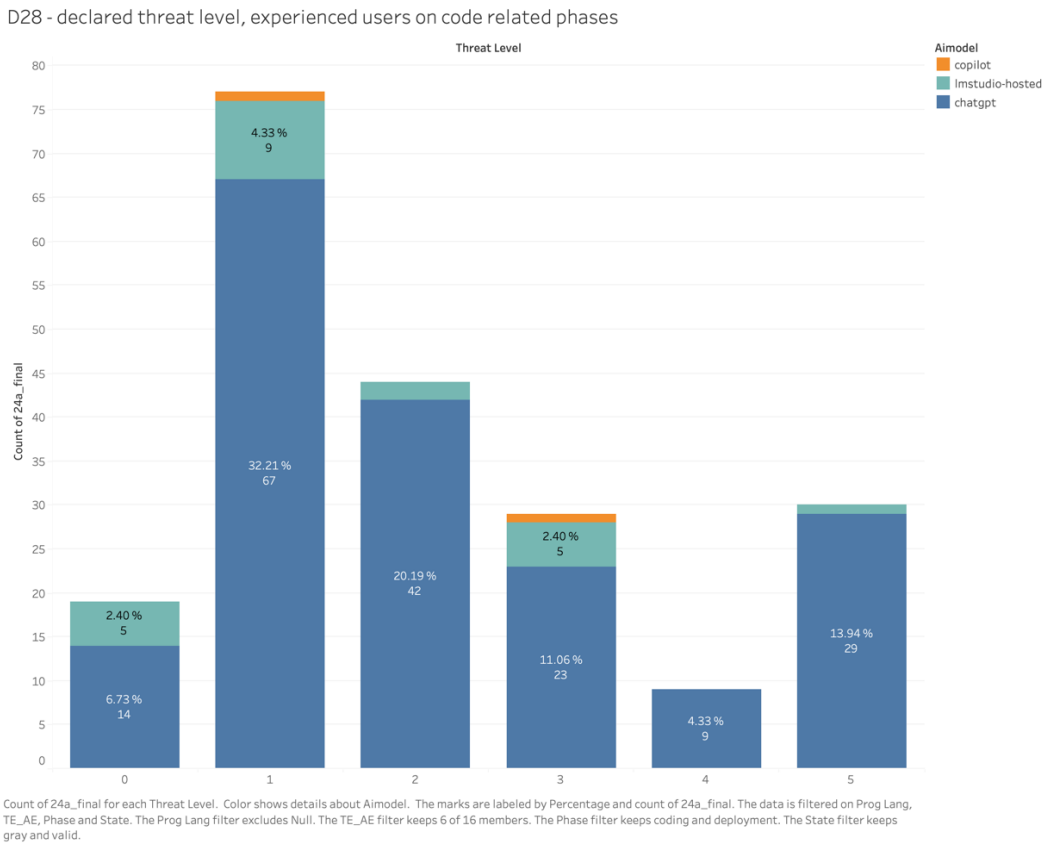

Figure 10.7.2 Declared threat level, experienced users on code-related phases (saas24a)

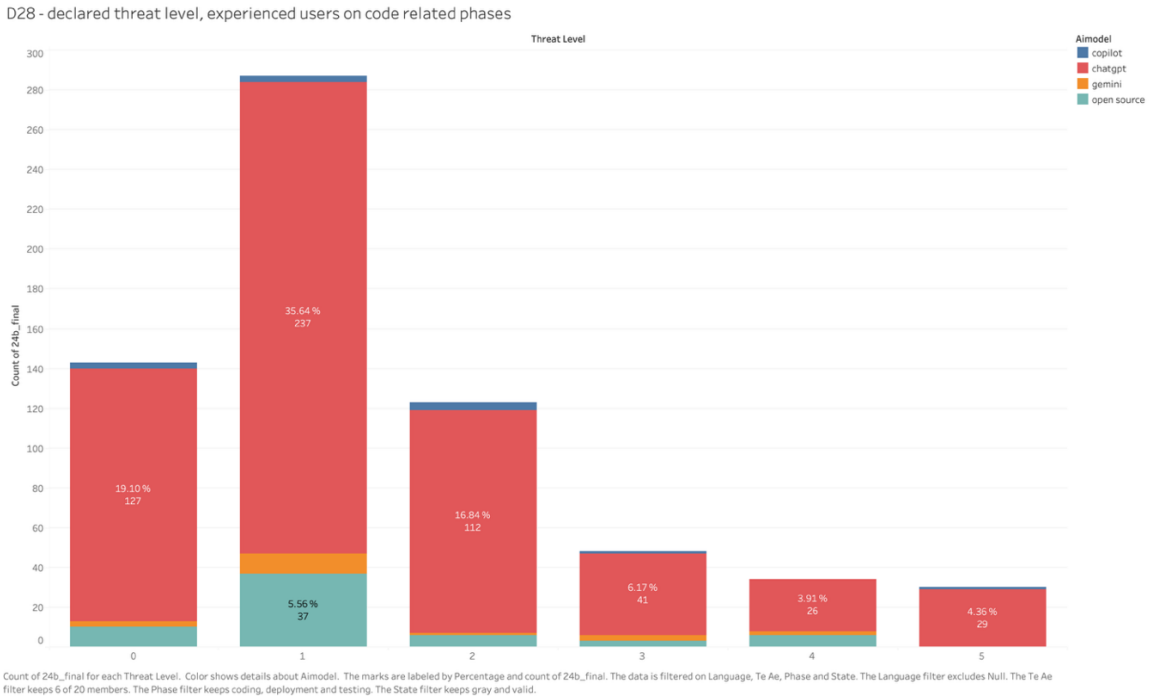

Figure 11.7.2 Declared threat level, experienced users on code-related phases (softeng24b)

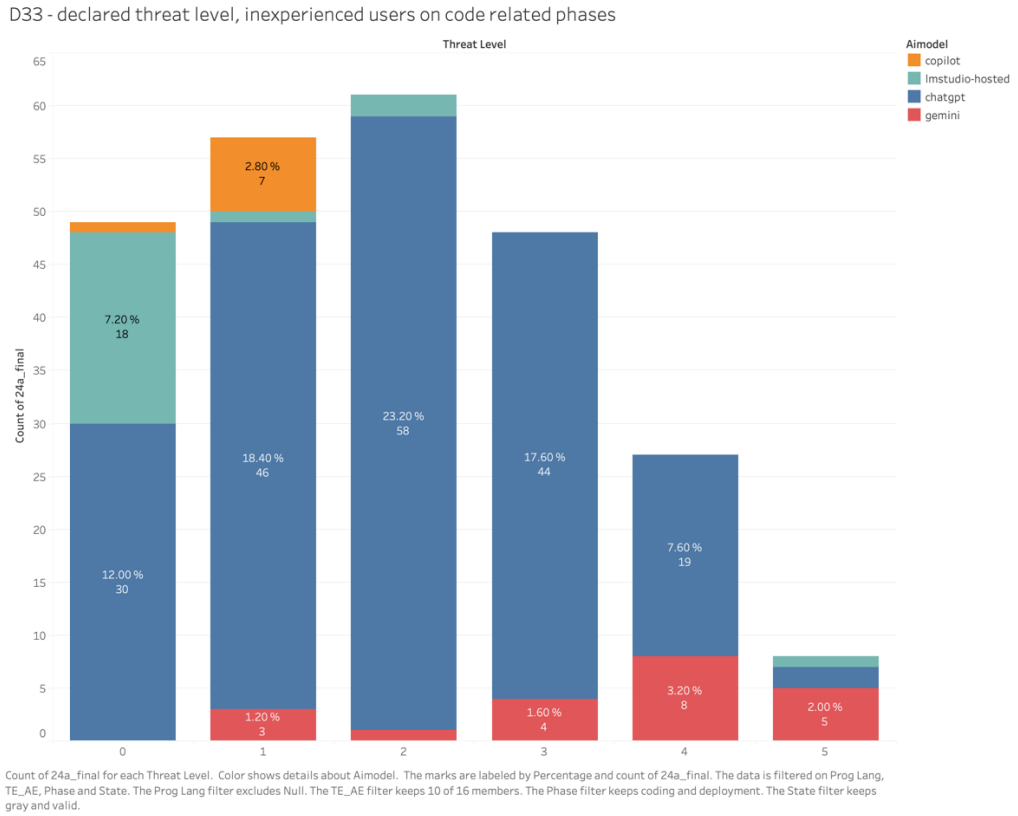

Figure 12.7.2 Declared threat level, inexperienced users on code-related phases (saas24a)

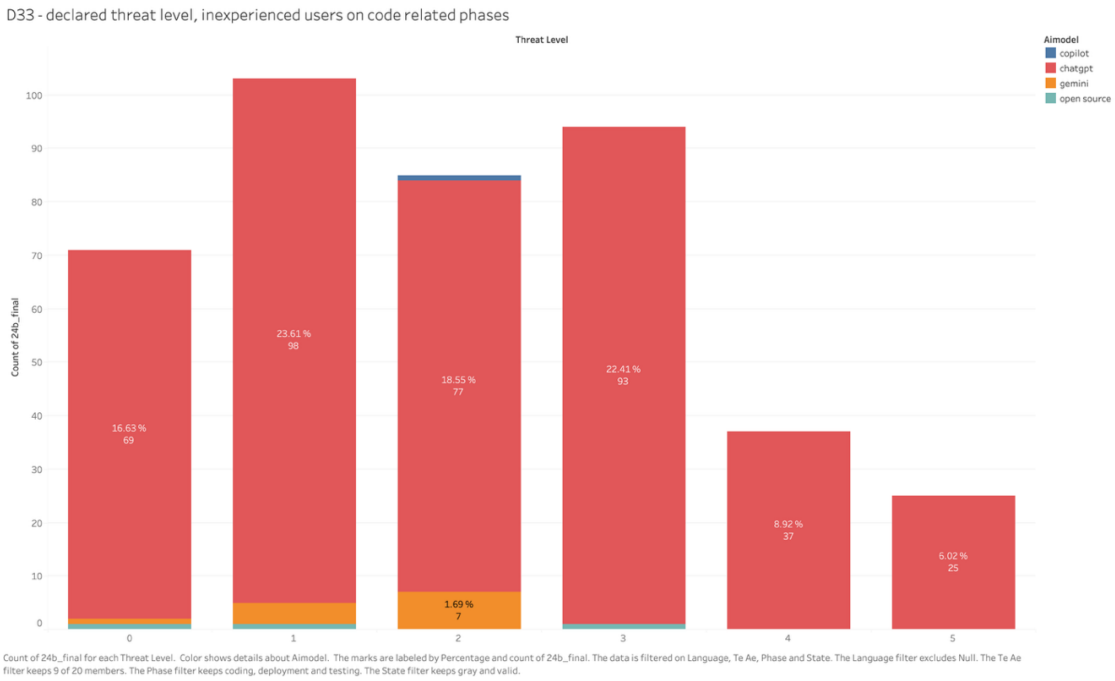

Figure 13.7.2 Declared threat level, inexperienced users on code-related phases (softeng24b)

8.3 Non-Code Phases

This section explores how students interacted with AI during the non-code related phases—that come before any code is written. These include requirements gathering, requirements specification, architecture planning, and system design. Unlike coding or testing, these early tasks are more abstract. They rely heavily on human interpretation, decision-making, and understanding, which makes AI support less predictable. Students often need help organizing ideas, sketching diagrams, or articulating goals—tasks that are open-ended by nature. Because of that, the alignment between what students expect and what the AI delivers is harder to pin down. This section looks at how AI performed in these early stages in terms of output quality, time savings, and emotional response, focusing on Research Questions RQ1, and RQ4.

Figures 14.7.3, 15.7.3, and 16.7.3 present student ratings of AI help quality in non-code related phases across the three course datasets. It's clear that among the conceptual phases, design receives the highest and most consistent ratings, followed closely by architecture. In all three datasets, student evaluations peak around "minor modifications" and "ready-to-use", or 3-4 in the numerically set databases, with very few reports of unusable output or lower numbered score. This suggests that AI tools perform well when students pose structured prompts involving diagram assistance, design alternatives, or architectural rationale templates. Worth mentioning is that architecture has an even distribution between minor or major modifications needed and ready-to-use quality in the dataset of SoftEng-23b, showing that AI was often helpful, its outputs in this phase required more careful interpretation in comparison to the better quality-provided prompts in design.

In contrast, requirements-related phases show more scattered and cautious evaluations. In SoftEng23b and SoftEng24b—the only two datasets where requirements gathering and specification were included—quality ratings for these phases flatten toward the center of the scale, with increased frequency of "minor modifications needed" or neutral responses (3-4). This reflects the interpretive ambiguity of the students in early-stage activities such as problem understanding, definition of functional or non-functional requirements or use case specifications. Unlike design and architecture, these phases lack a definite structure and involve mainly on a more abstract answer choice. AI tools, when prompted with open-ended instructions like "generate user requirements" or "suggest non functional requirements", tend to produce generalized templates or speculative assumptions that often require extensive human revision.

Despite this variability in perceived quality, time-related metrics paint a more uniformly positive picture. Figures 17.7.3, 18.7.3, and 19.7.3 show the average time allocated and average time saved per non-code phase across the three datasets. In SoftEng-23b, the requirements specification stands out since the average time allocated by the students in this phase had an average of 0.93 hours but the estimate time saved was 2.26, suggesting that AI was particularly useful in structuring the specifications in which the users may not

have had the knowledge to complete or organize. In SoftEng-24b, requirement gathering showed a similar but higher ratio, with students allocating an average of 0.59 and saving 1.52—nearly three times the time allocated. These results imply that even when AI outputs weren't rated perfect and even with major modifications, they provided enough direction that still saved a lot of time for the students.

This difference between declared output quality and strong time savings highlights an important finding. Even when the AI doesn't deliver perfect results, it often helps students get started faster. Drafting a paragraph, outlining an idea, or suggesting a template—these seamlessly minor contributions reduce the time load of early-stage work. Students in these early stages receive AI's suggestions, rewrite what doesn't fit, and move forward. The AI acts as a productive idea provider, especially in phases where the challenge is figuring out how to begin when definitions are unfamiliar.

When it comes to how students feel about AI in these early, non-code related phases, the emotional landscape is more subtle than in code related phases. Figures 20.7.3 and 21.7.3 show the declared threat levels from experienced users working in requirements, architecture, and design. In SaaS-24a (Figure 20.7.3), responses are fairly balanced, with a normal-like distribution centered around levels 2–3. This suggests a moderate, thoughtful stance—these students don't see AI as a major threat in conceptual phases, but they're not completely unconcerned either. There's a measured awareness that while AI can help with idea organization or structural planning, the final judgment still rests with the human developer.

In SoftEng-24b (Figure 21.7.3), however, the responses skew more strongly toward the low end of the scale. Most experienced students in these phases place their threat rating at 1, showing a clear relief that AI will not be able to replace these actions. This shift suggests growing confidence in treating AI as a brainstorming or drafting assistant in the early stages of development. Additionally, the fact that SaaS-24a excludes the phases of requirements could indicate that the requirement phases is the reason why SoftEng-24b is more skewed to the left, thus there is no threat of takeover of AI in the requirement stages yet.

Inexperienced users tend to show even less concern. In SaaS-24a (Figure 22.7.3), the distribution is slightly skewed left, with most students clustering around levels 1 and 2. These users still show some caution, but it's less. They appear comfortable using AI as a support mechanism, especially in non-code related tasks where the risk of error is low and the main challenge is getting started.

The SoftEng24b data for inexperienced users (Figure 23.7.3) shows an even stronger skew to the left even in comparison to the same dataset but with experienced users, with most responses sitting between 0 and 2—and especially concentrated at 0 and 1. For this group, AI is not seen as a threat at all. It's viewed almost entirely as a helpful presence—something that assists with ideation, brainstorming, guides structure or defines unknown terminology

at the start of a task. The lack of concern reflects a high level of comfort with letting AI contribute in areas where interpretation, rather than precision, is what matters most.

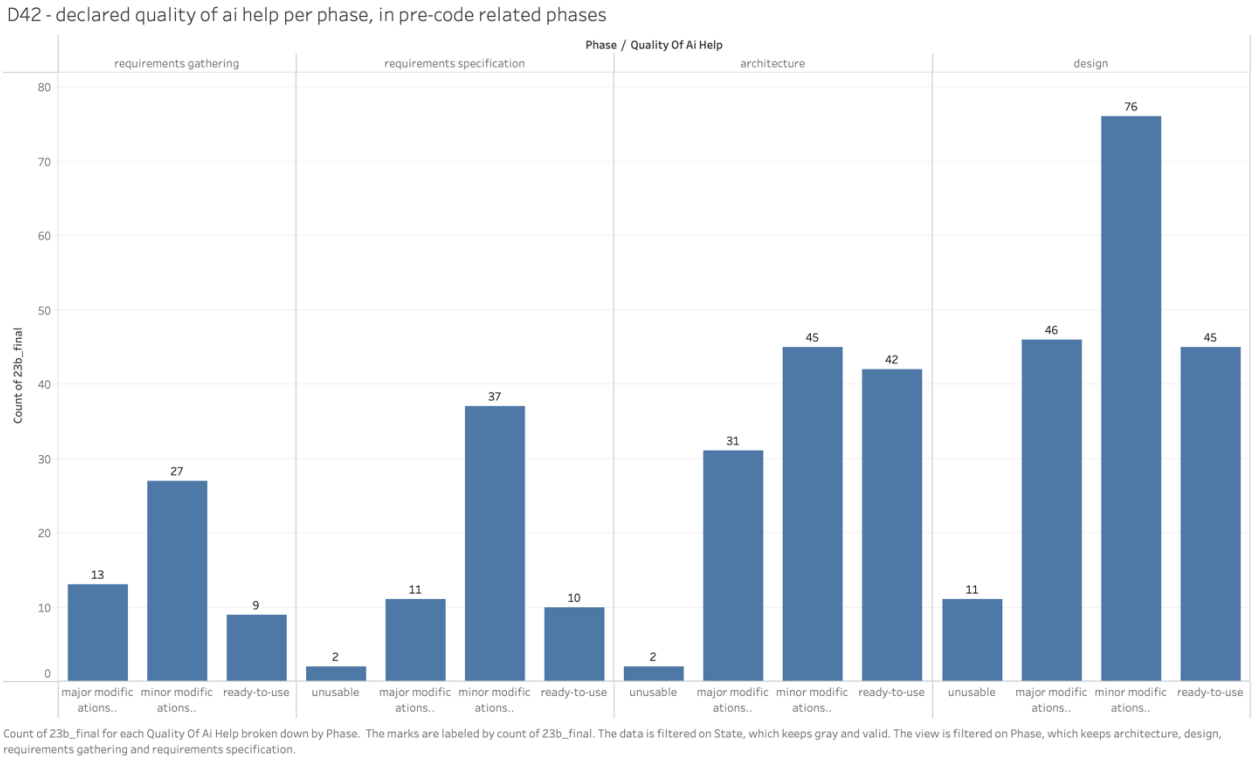

Figure 14.7.3 Declared quality of AI help per phase in non-code phases (softeng23b)

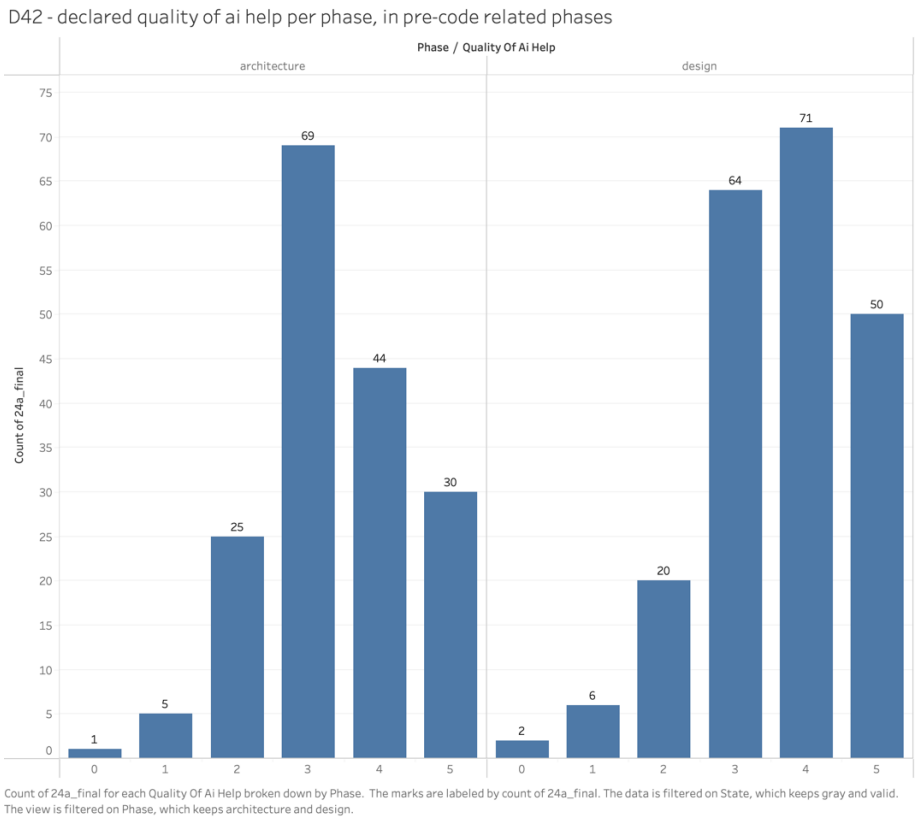

Figure 15.7.3 Declared quality of AI help per phase in non-code phases (saas24a)

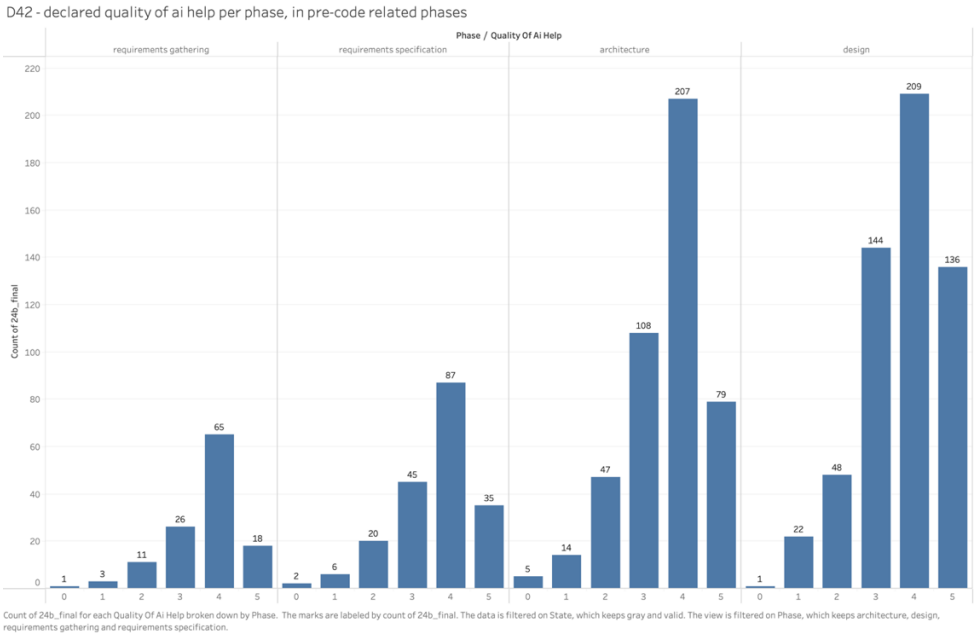

Figure 16.7.3 Declared quality of AI help per phase in non-code phases (softeng24b)

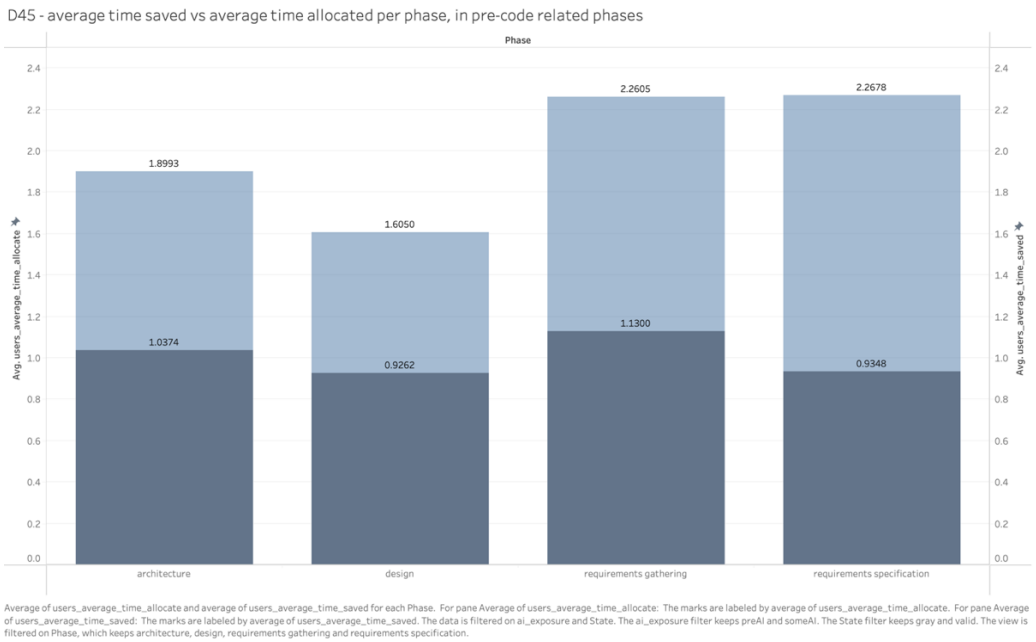

Figure 17.7.3 Average time saved vs average time allocated per phase in non-code phases (softeng23b)

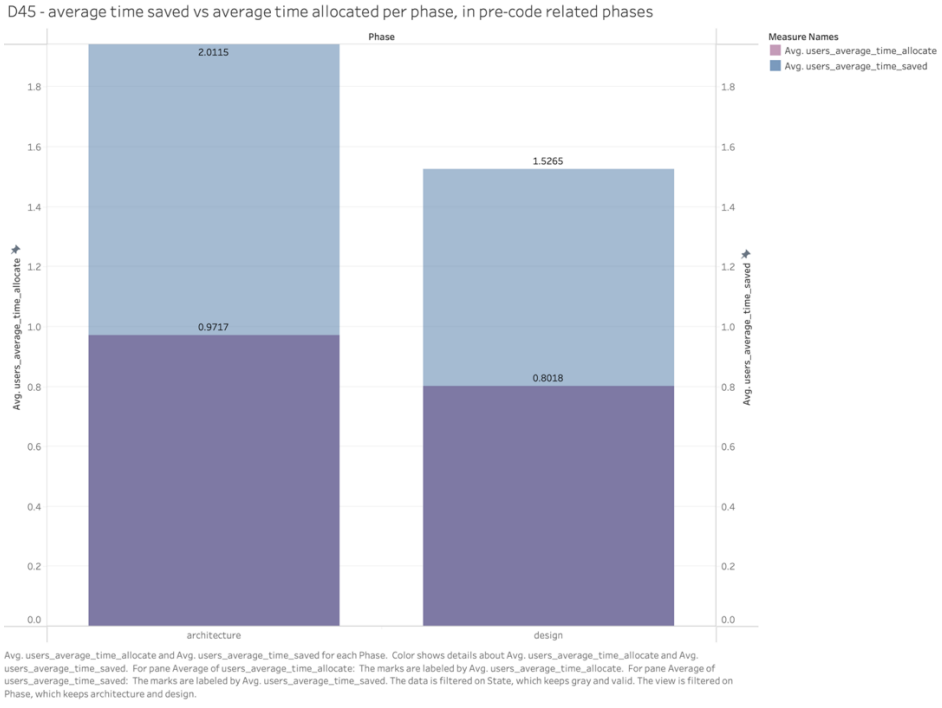

Figure 18.7.3 Average time saved vs average time allocated per phase in non-code phases (saas24a)

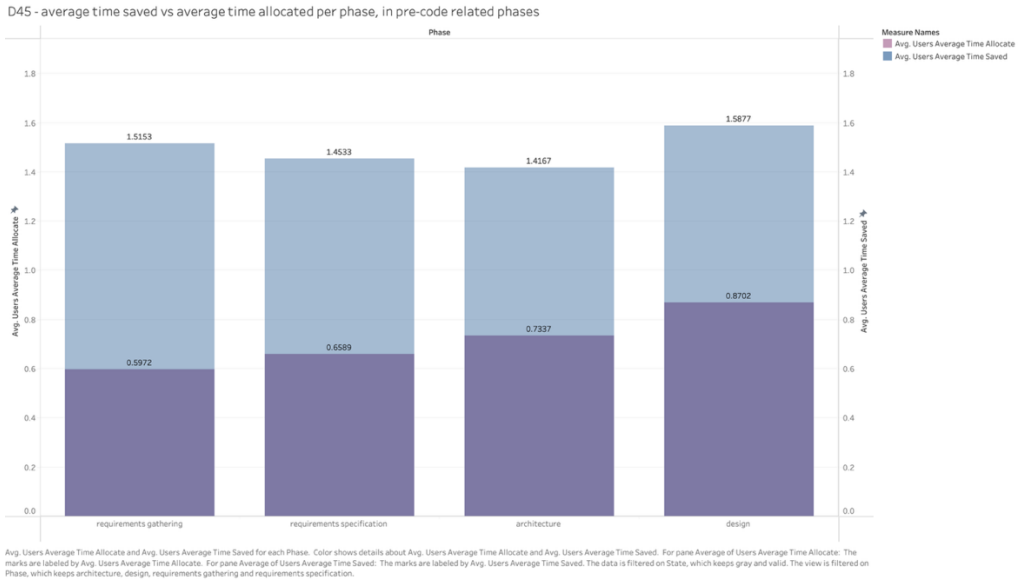

Figure 19.7.3 Average time saved vs average time allocated per phase in non-code phases (softeng24b)

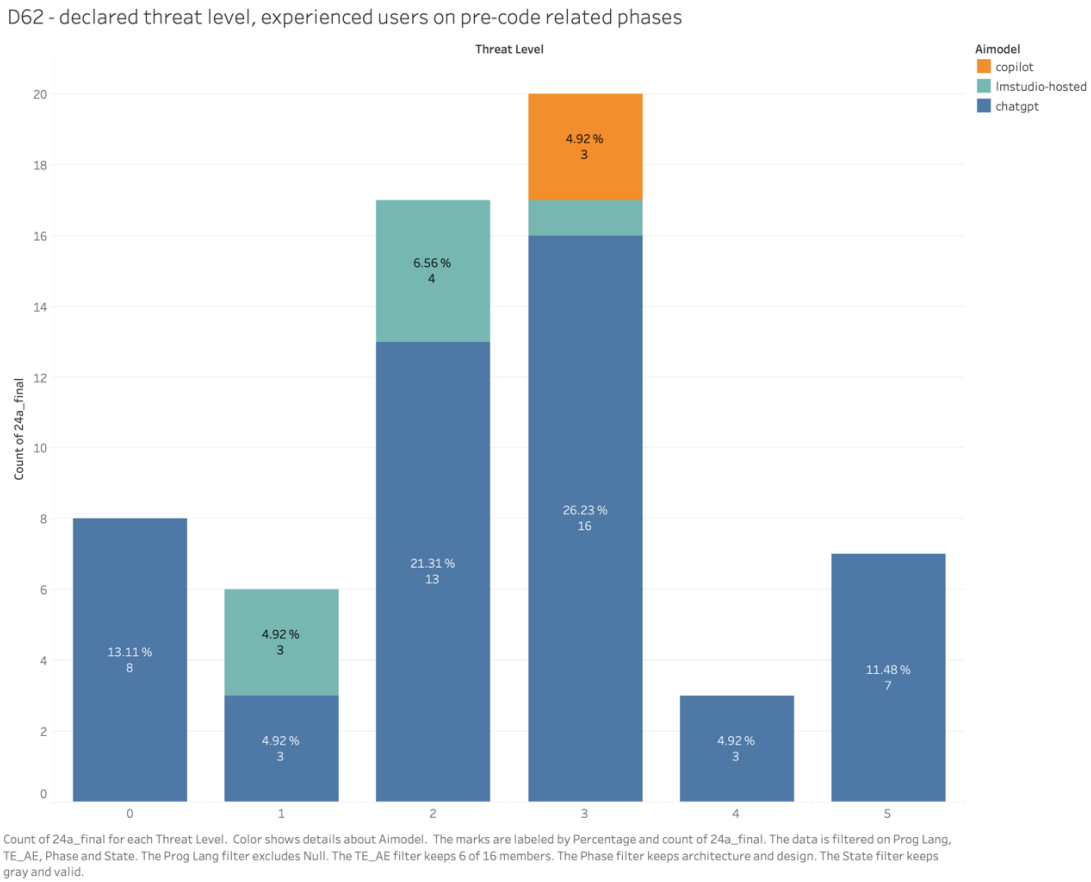

Figure 20.7.3 Declared threat level, experienced users on non-code phases (saas24a)

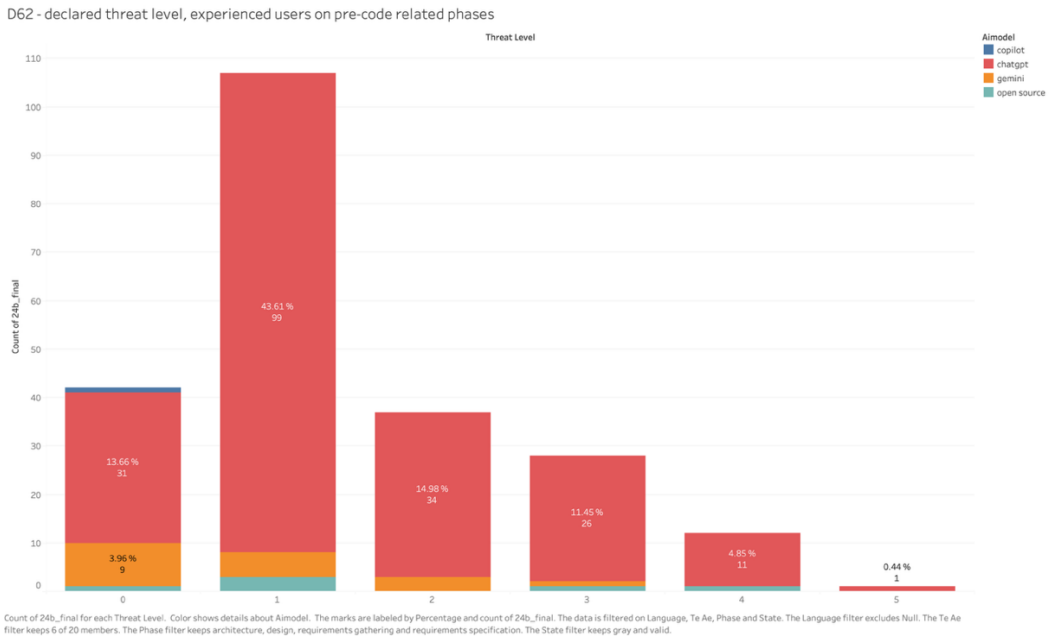

Figure 21.7.3 Declared threat level, experienced users on non-code phases (softeng24b)

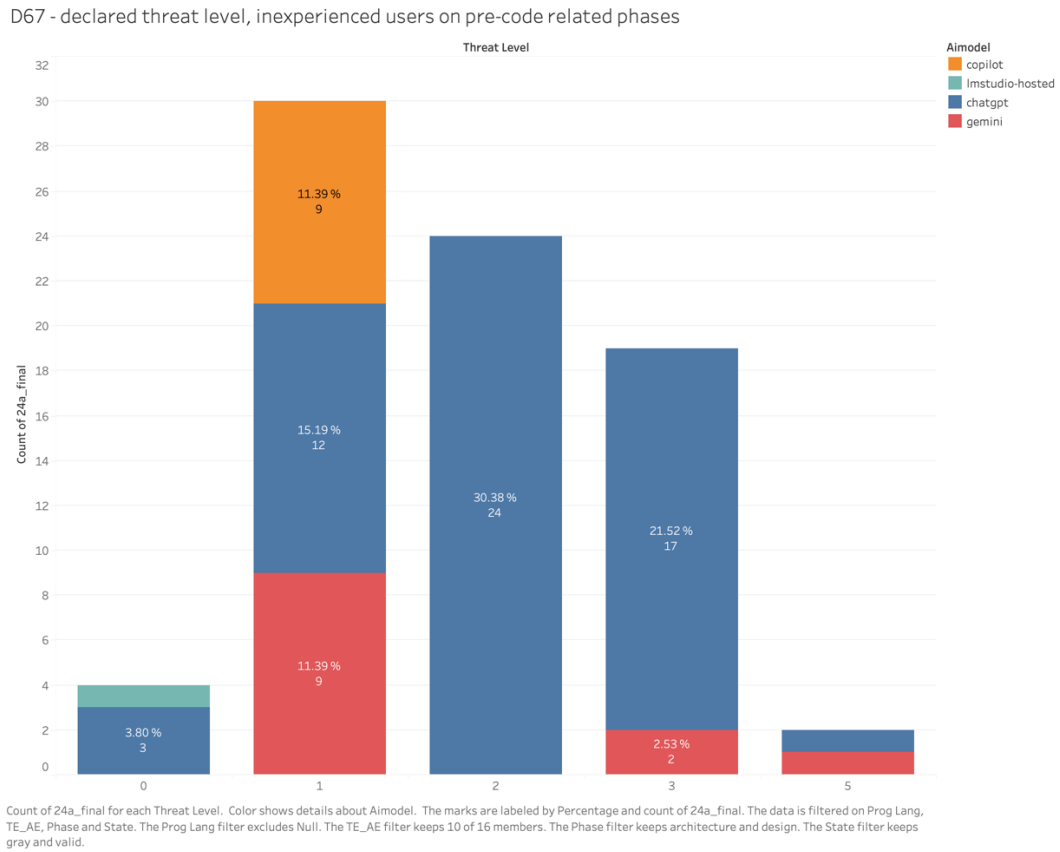

Figure 22.7.3 Declared threat level, inexperienced users on non-code phases (saas24a)

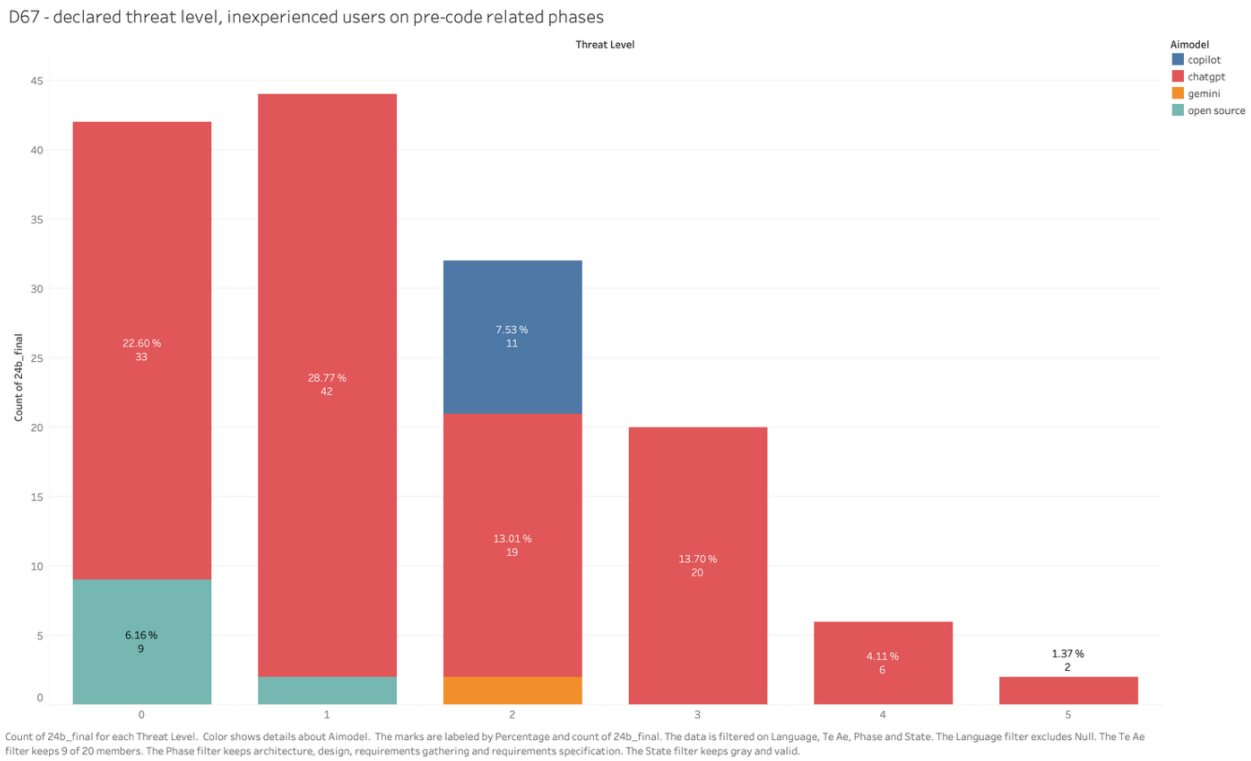

Figure 23.7.3 Declared threat level, inexperienced users on non-code phases (softeng24b)

8.4 Students who took part in Both SoftEng-23b and SaaS-24a

This section analyzes variation in AI perception among 51 students who submitted structured data in both the SoftEng-23b and SaaS-24a courses. These repeated participants allow for within-subject comparison across time, enabling direct evaluation of how declared experience, emotional stance, and AI-related judgments change through consecutive academic courses. The figures referenced track changes in user-reported metrics from the first to the second course and are used to address Research Questions RQ4 and RQ5.

Figure 24.7.4 shows how students rated their experience with AI tools across the two courses. The results shift clearly to the right: in SoftEng-23b, most students placed themselves around levels 2–3 (moderate familiarity of "some" and "enough"), while in SaaS-24a, most responses fall between 3 and 4. Notably, no students rated themselves at the lowest levels (0–1) in the second course, meaning everyone felt at least somewhat confident using AI after an entire semester. This shift isn't just about using AI more often—it reflects that students learned how to use it more effectively, not just sending prompts without preprocessing them first.

A similar trend appears in Figure 25.7.4, which looks at how confident students declared they were in the said action. Although the increase is more gradual, there's still a clear move upward. Students report more comfort with the tasks themselves, suggesting that AI didn't just make things faster—it helped them build skill and experience by giving them feedback, showing patterns, and encouraging clearer thinking. Together, the two figures show that students improved in both tool use and action execution, gaining confidence on both fronts.

Figures 26.7.4 and 27.7.4 capture current emotional stance—"generic feeling now"—toward AI tools in SaaS-24a, for the students who were marked as experienced or inexperienced based on their declaration in the first course. Both groups—those who were experienced and those who were inexperienced—report neutral to positive feelings (modal values around 3–4), indicating convergence in present-time trust. However, future-oriented optimism shows a more dramatic skew. In Figures 28.7.4 and 29.7.4—the figures that display the distribution of "generic feeling future", the distributions tilt heavily toward the maximum value (5), with both groups indicating strong confidence in the future role of AI tools. This shift is especially shown among initially inexperienced users, who enter the second course not only with technical improvements and more knowledge but also with more assurance that AI tools will remain relevant and increasingly useful.

Learning outcomes—knowledge acquired—are evaluated in Figures 30.7.4 and 31.7.4. These figures track declared knowledge acquired from AI-assisted tasks in SaaS-24a. Both groups report values centered around level 3, with modest right skew. Inexperienced users show slightly stronger clustering around higher values, which may be attributed to steeper learning curves. These results suggest that students perceive AI tools as having meaningful and learning utilities—even though they were not formally framed as teaching tools. This

internalized attribution of learning benefit reinforces the role of AI as a learning assist in software development education.

Finally, perceived output quality is examined in Figures 32.7.4 and 33.7.4. Both experienced and inexperienced students report strong AI output quality, with dense distributions between levels 3 and 5. Inexperienced users, report that the AI provided better quality than what the experienced users reported. This could suggest that the inexperienced users are claiming the AI output just because it works—if its code related—or because they think its correct so they accept is as good quality without knowing if it's valuable or not.

In conclusion, the data from the students who took both the SoftEng-23b and SaaS-24a reveal a clear progression in how AI is perceived and used over time. Confidence in both tool and action use increased dramatically. Emotional responses also shifted through time—from cautious and average rating to strongly optimistic about future use of AI by both the experienced and inexperienced groups of students. Inexperienced users, particularly, showed growth in both trust and perceived knowledge acquired thus seemed to use AI for learning and understanding. The experienced users on the other hand, remained more measured in their initial evaluations, however we still notice an improvement of knowledge acquired. These shifts reflect more than just familiarity over time—they show that students grow in their AI interactions when given the space to engage more with it—as they did during the developmental cycle of both projects.

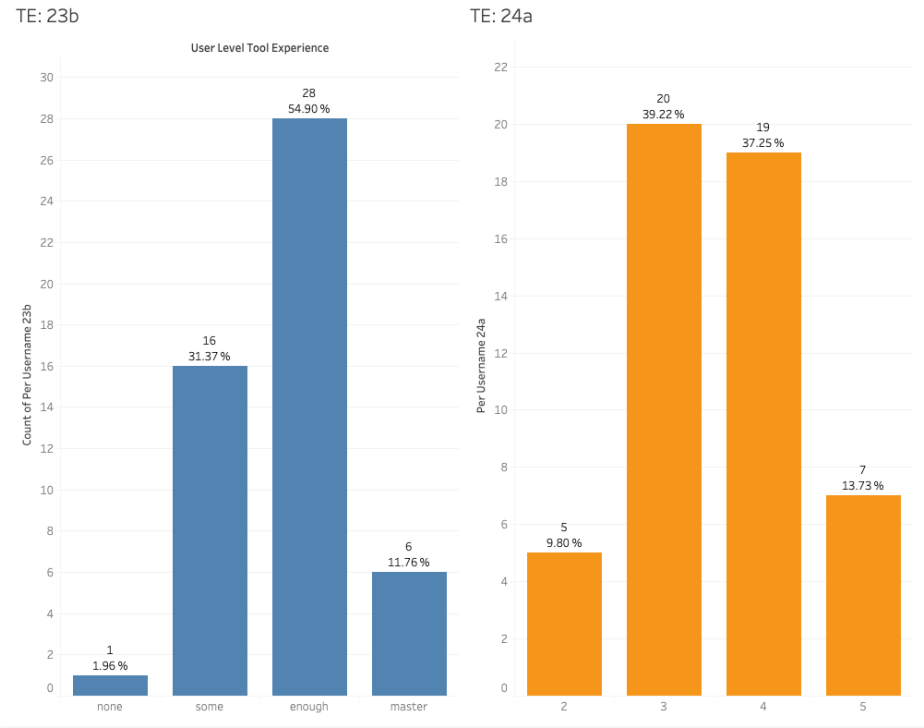

Figure 24.7.4 Declared tool experience over both years (softeng23b → saas24a)

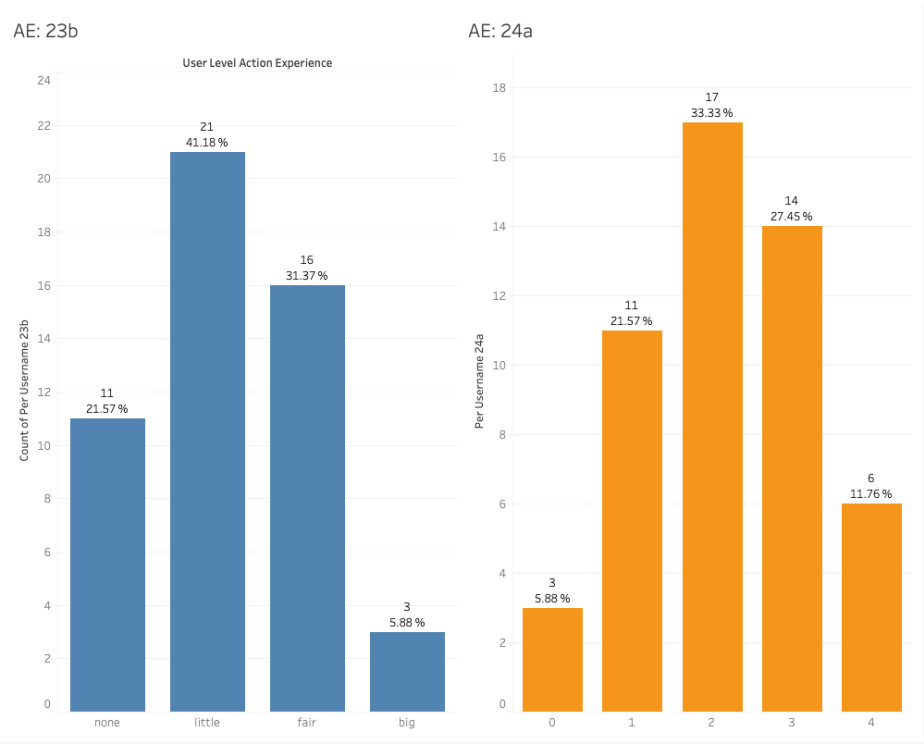

Figure 25.7.4 Declared action experience over both years (softeng23b → saas24a)

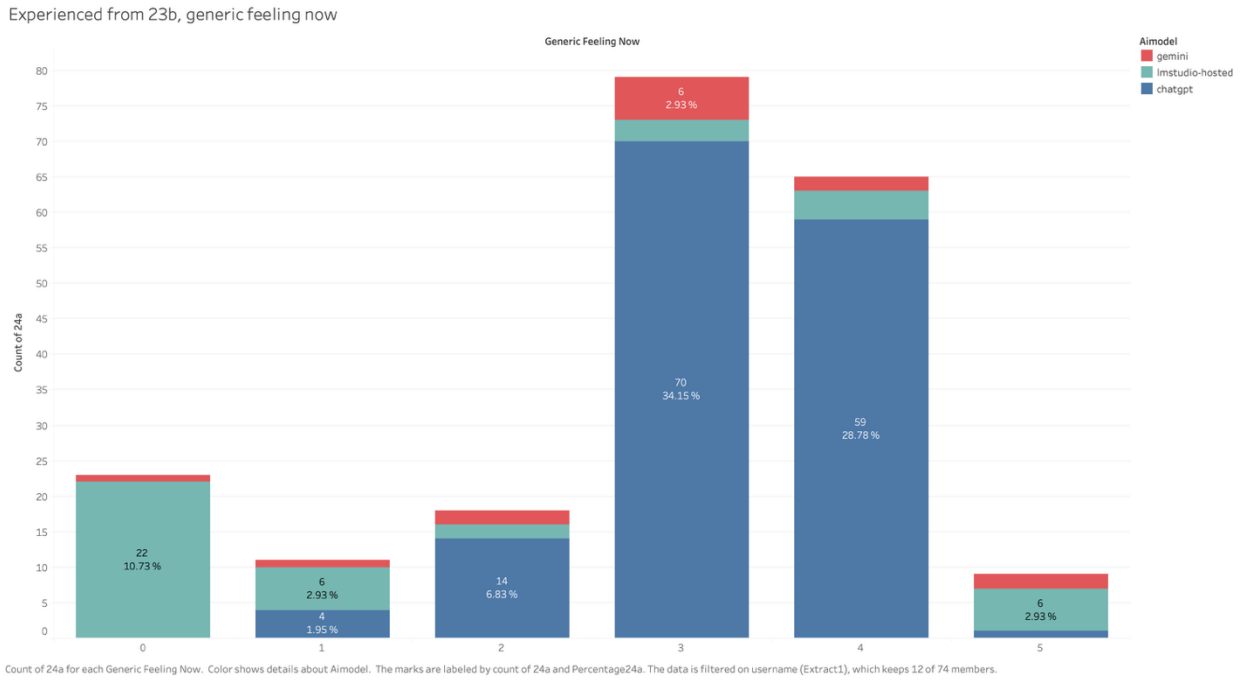

Figure 26.7.4 Declared current feeling (now), experienced users from softeng23b on saas24a responses

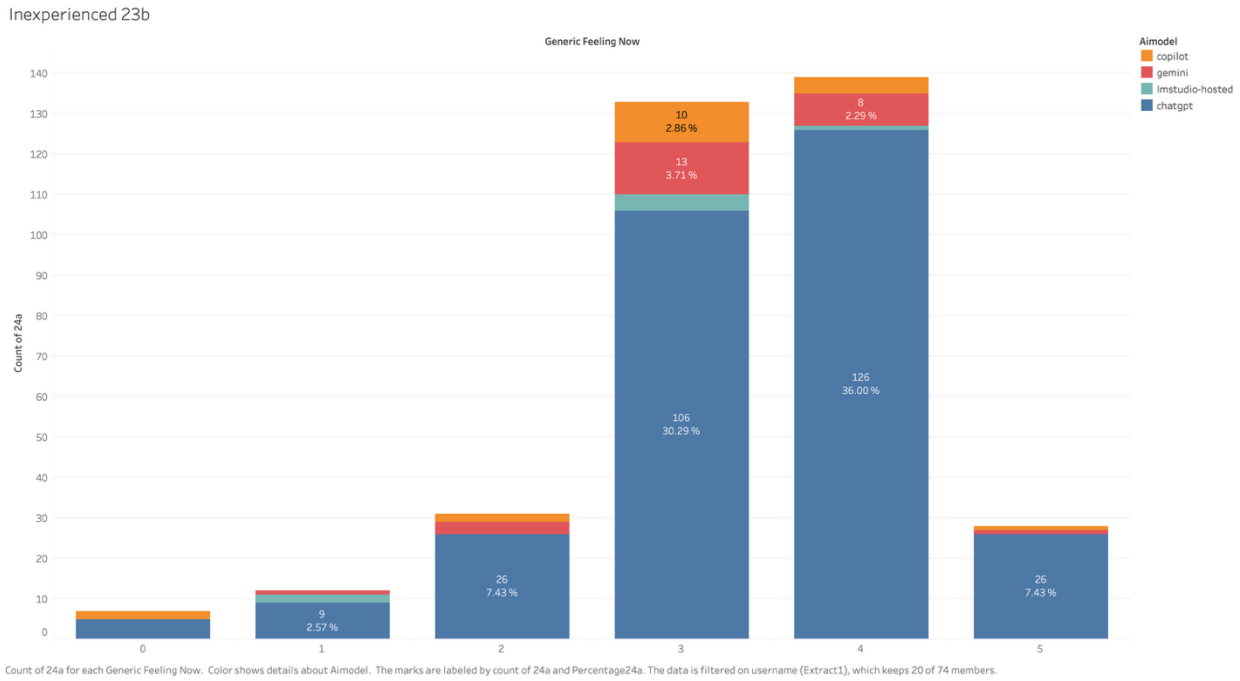

Figure 27.7.4 Declared current feeling (now), inexperienced users from softeng23b on saas24a responses

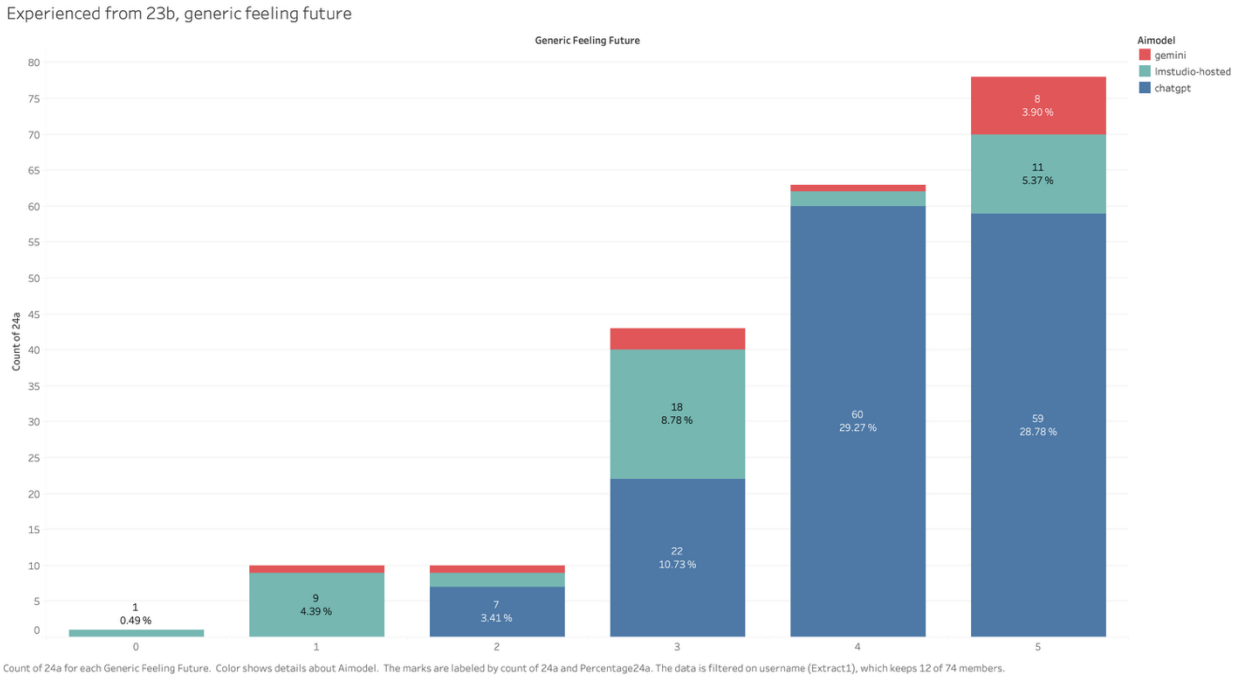

Figure 28.7.4 Declared future feeling, experienced users from softeng23b on saas24a responses

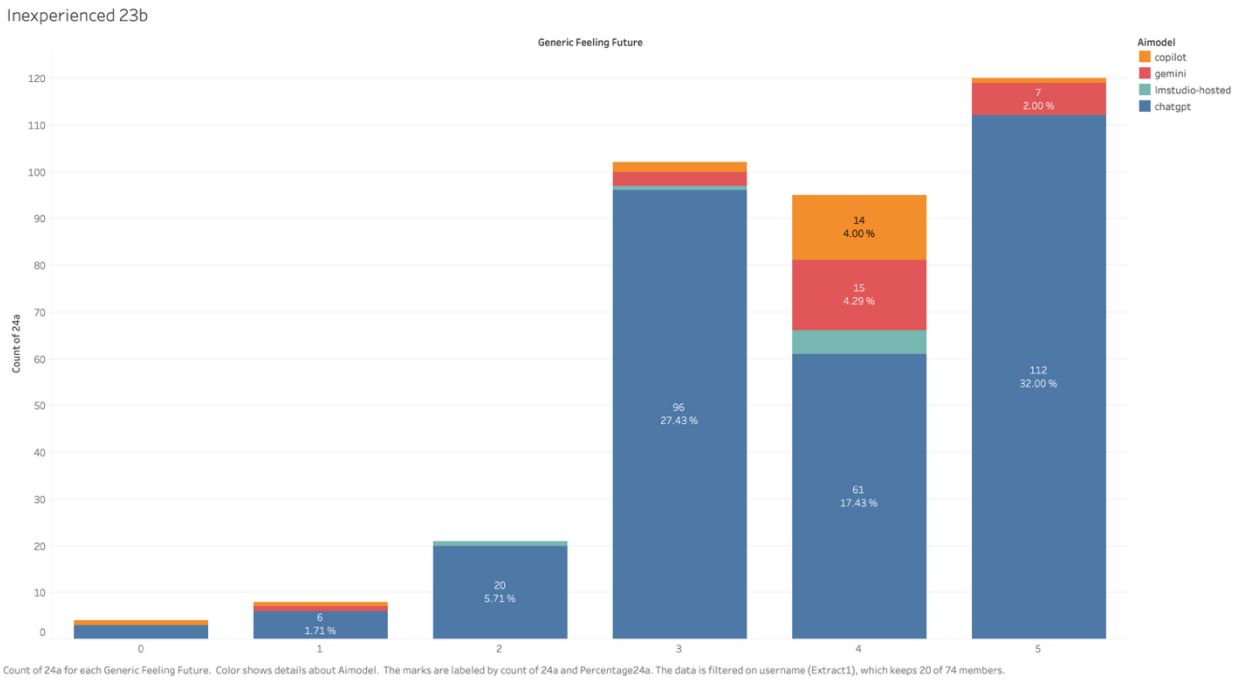

Figure 29.7.4 Declared future feeling, inexperienced users from softeng23b on saas24a responses

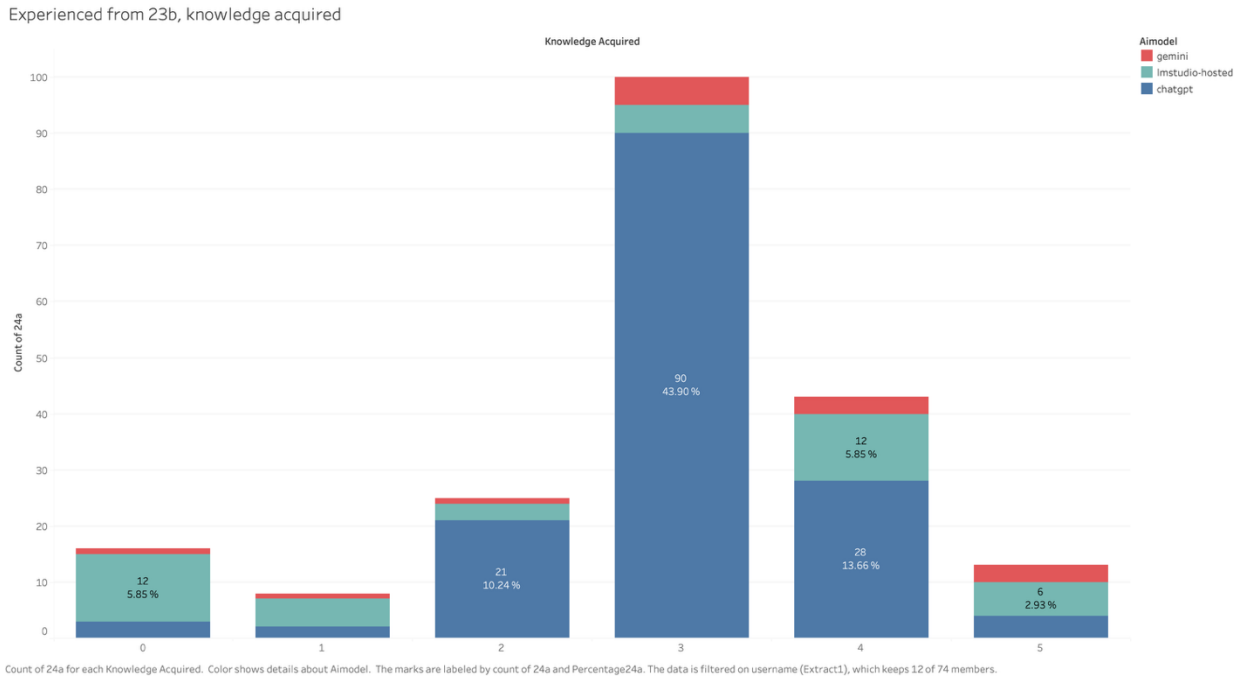

Figure 30.7.4 Declared knowledge acquired, experienced users from softeng23b on saas24a responses

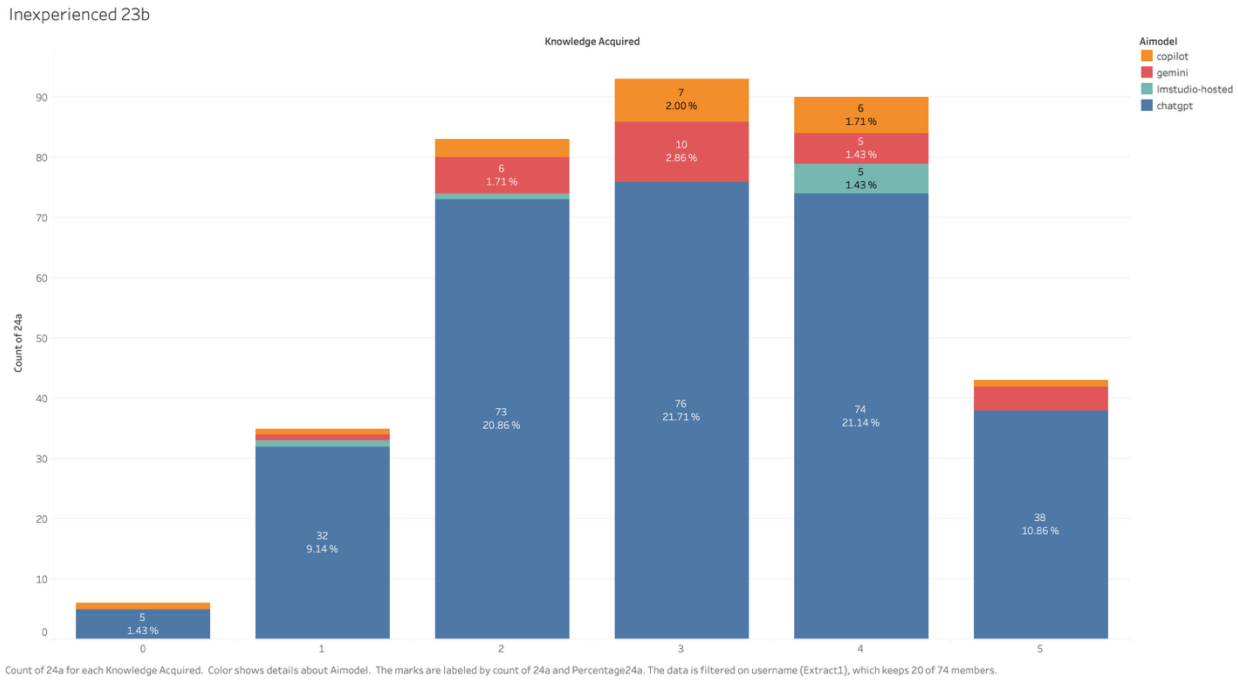

Figure 31.7.4 Declared knowledge acquired, inexperienced users from softeng23b on saas24a responses

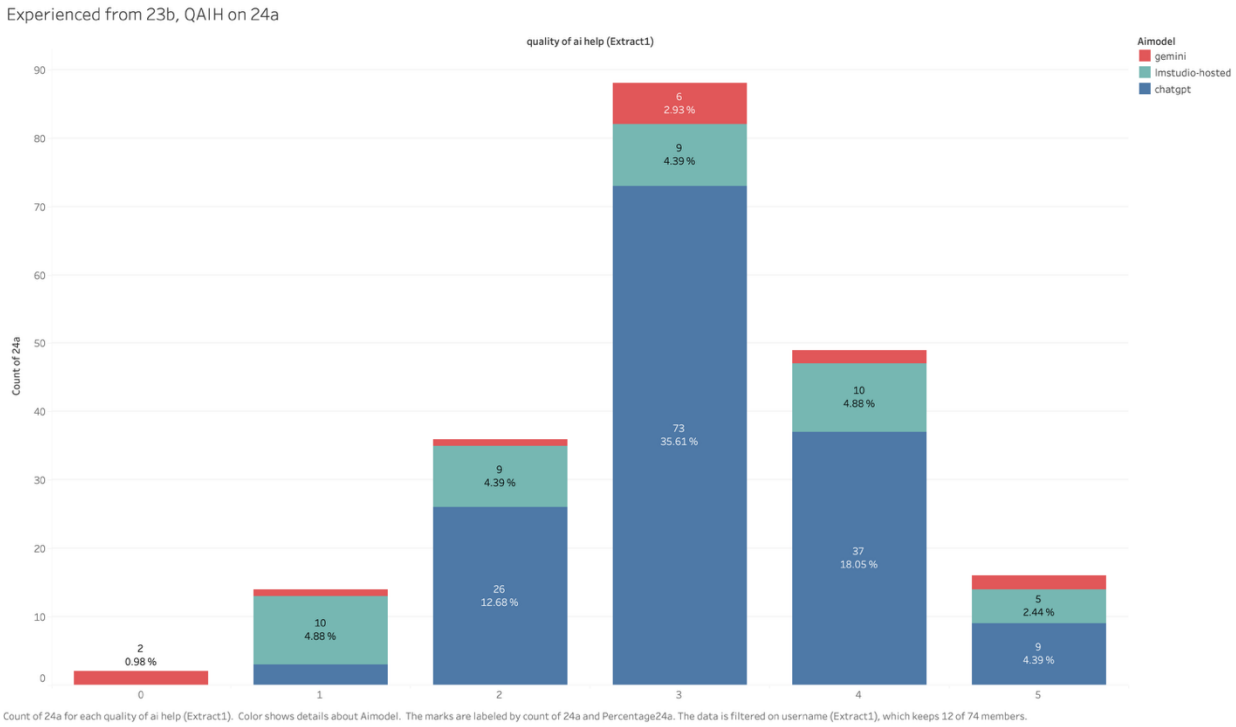

Figure 32.7.4 Declared quality of AI help, experienced users from softeng23b on saas24a responses

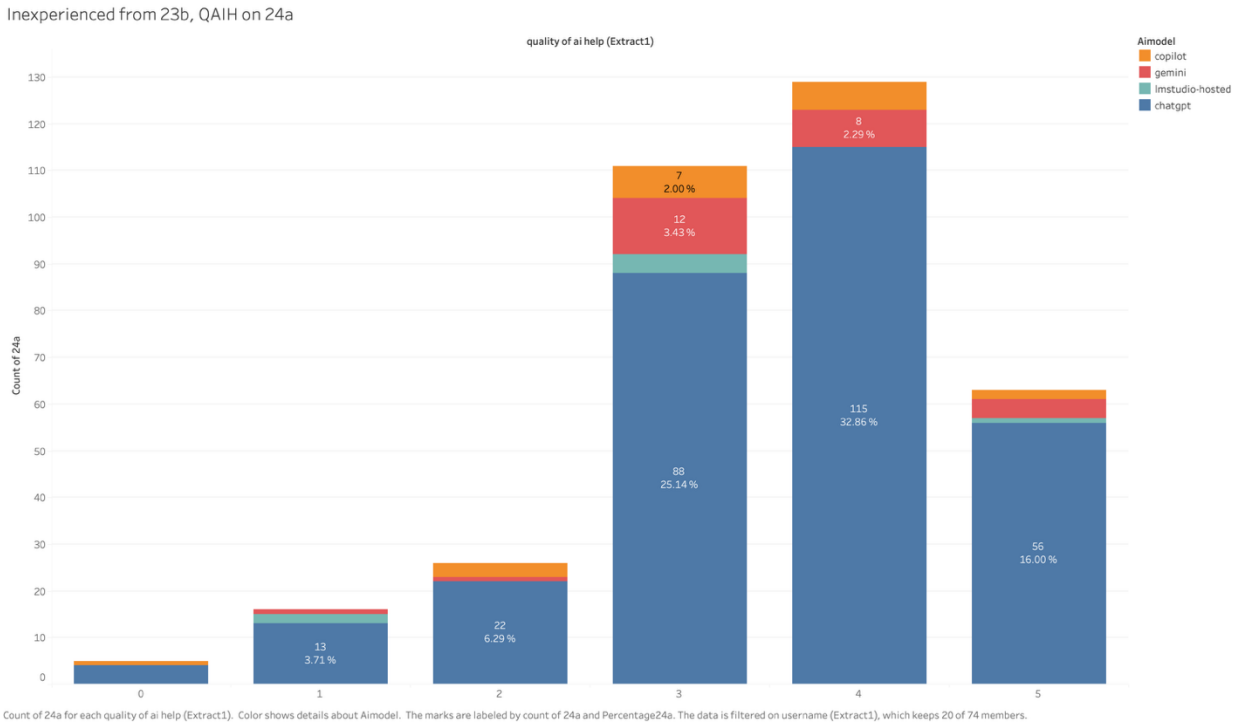

Figure 33.7.4 Declared quality of AI help, inexperienced users from softeng23b on saas24a responses

8.5 Programming Language and Experience Effects

This section explores how the programming language in use shaped students' perceptions of AI-generated output quality, while also revealing how experience levels influenced those judgments. It directly answers Research Question RQ3, which asks whether programming language affects perceived AI utility and quality in output, and informs RQ4, which considers how user experience modifies that perception.

Figures 34.7.5, 35.7.5, and 36.7.5 show that across all three courses, Python and JavaScript clearly led the way in terms of positive student evaluations. Most submissions involving these two languages were rated at level 4-5 or as—"minor modifications needed" or "ready-to-use"—with few responses falling below that. Python, in particular, stands out—likely due to its wide adoption in backend development, scripting, and AI assignments. JavaScript, along with Node.js and related tools, also performed strongly, especially in frontend and CLI-related tasks where students often relied on AI to start building components from scratch, generate handlers, or polish UI logic. Languages like SQL and YAML tell a different story—not because they were rated poorly, but because they were used far less frequently. These languages appeared in the data much less often, making it difficult to draw strong conclusions about quality. Where entries did exist, the distribution of quality ratings was similar to other languages, but the sample size was noticeably smaller. This may be due to the fact that the AI interaction was quick, clear-cut and ready-to-use, thus the students decide not to log their interaction.

Figures 37.7.5 through 39.7.5 focus on experienced users, who gave even stronger approval to the quality of AI provided. In these cases, the vast majority of entries were rated 4 or 5, suggesting that experienced students not only knew how to prompt effectively but also had a better sense of what to expect from the tool. Their use of AI was more efficient—they used it where it worked well, and avoided it in places where it didn't. Experienced students seemed to rely on AI most in languages where they trusted the model's reliability and knew how to fix issues when and if needed, also where some modifications were needed clearly indicates that the user has the necessary knowledge to fix the issues that arised.

In Figures 40.7.5 to 42.7.5, we see the responses from less experienced users. Most students gave high marks, with the majority of responses landing at 4-5. Compared to experienced users, inexperienced users had a noticeable number of responses as 3. This could suggest that even when AI produced structurally correct or mostly functional code, some students didn't recognize it as such and believed it needed more modification than it actually did. Without the experience to alter, diagnose or trust the code provided they could ask the prompt to provide the entire file—where in cases where the file is 300 lines it makes the LLM overwhelm leading to incomplete or wrong output.

In conclusion, programming language and user experience both played a significant role in how students perceived the quality of AI-generated code. Python and JavaScript

consistently received the highest ratings and most submissions, especially among the experienced users who knew how to prompt more effectively and modify the output when needed. Inexperienced users also rated these languages highly, but their lower confidence was evident in more frequent mid-level scores, likely showing their uncertainty than actual output errors. Languages who had lower number of submissions recorded, were likely underreported due to the students' quick ai interaction, and the judgment of the students to not log the AI interaction. The data shows that students perception of AI quality is shaped not only by the language used but also by the students ability to understand, rewrite, and evaluate the code provided.

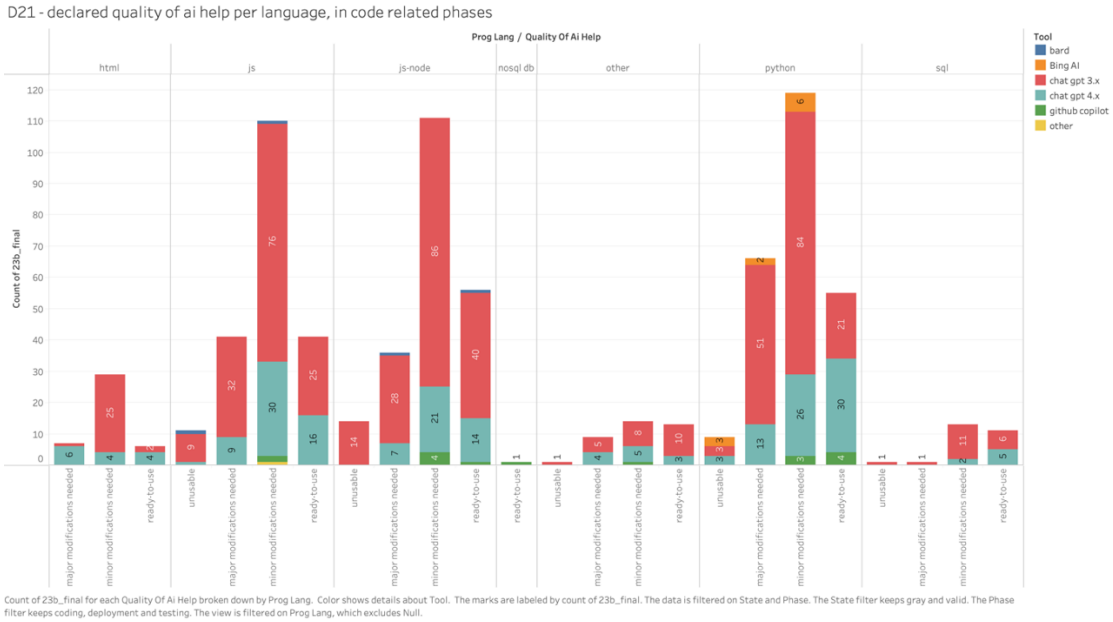

Figure 34.7.5 Declared quality of AI help by programming language (softeng23b)

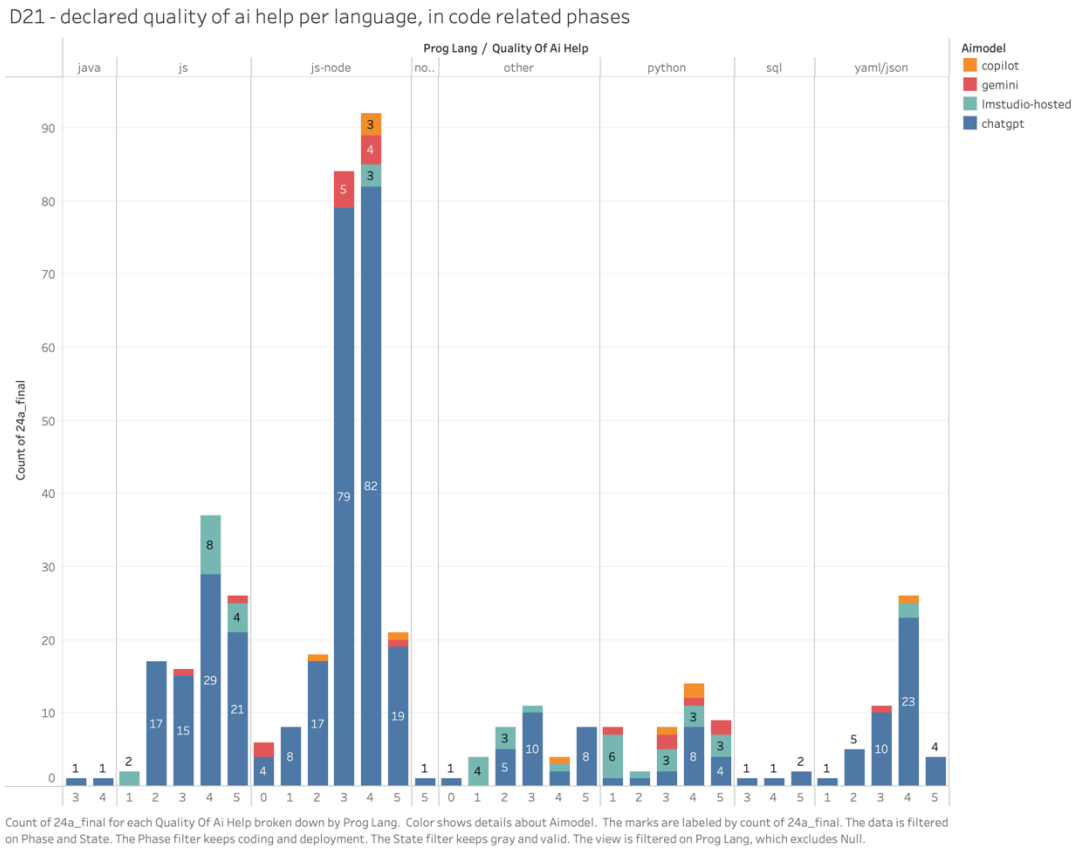

Figure 35.7.5 Declared quality of AI help by programming language (saas24a)

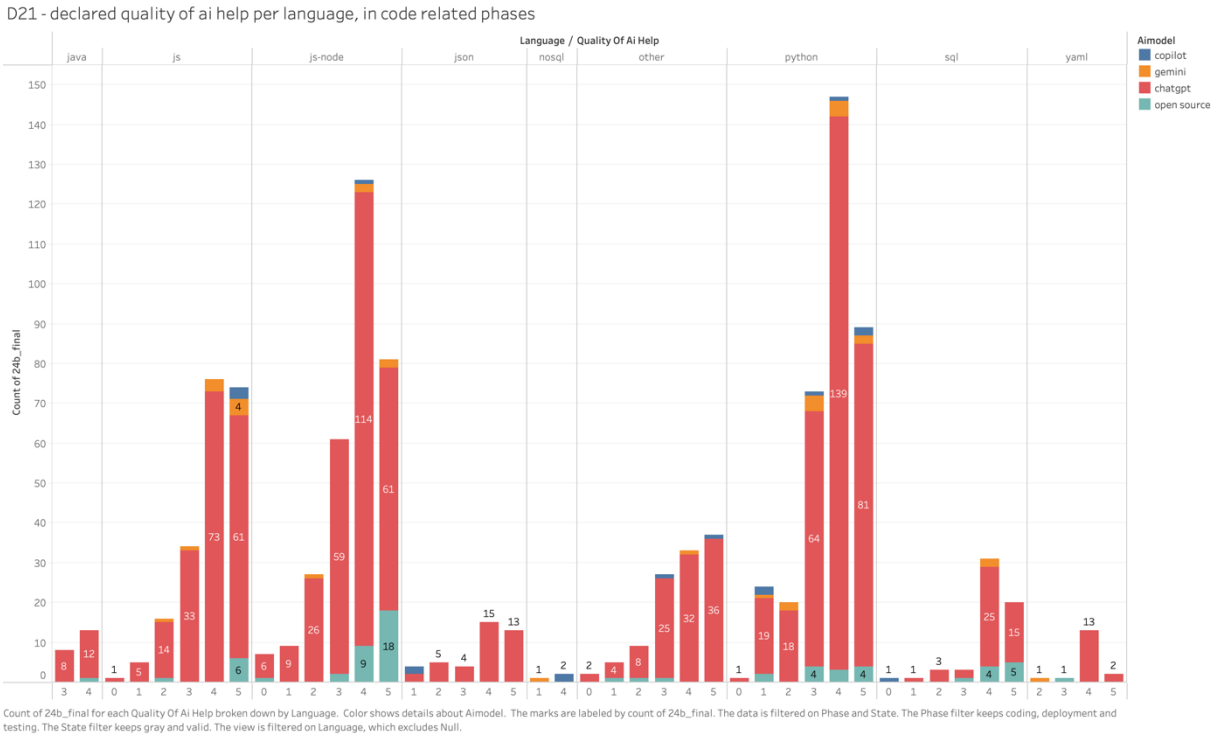

Figure 36.7.5 Declared quality of AI help by programming language (softeng24b)

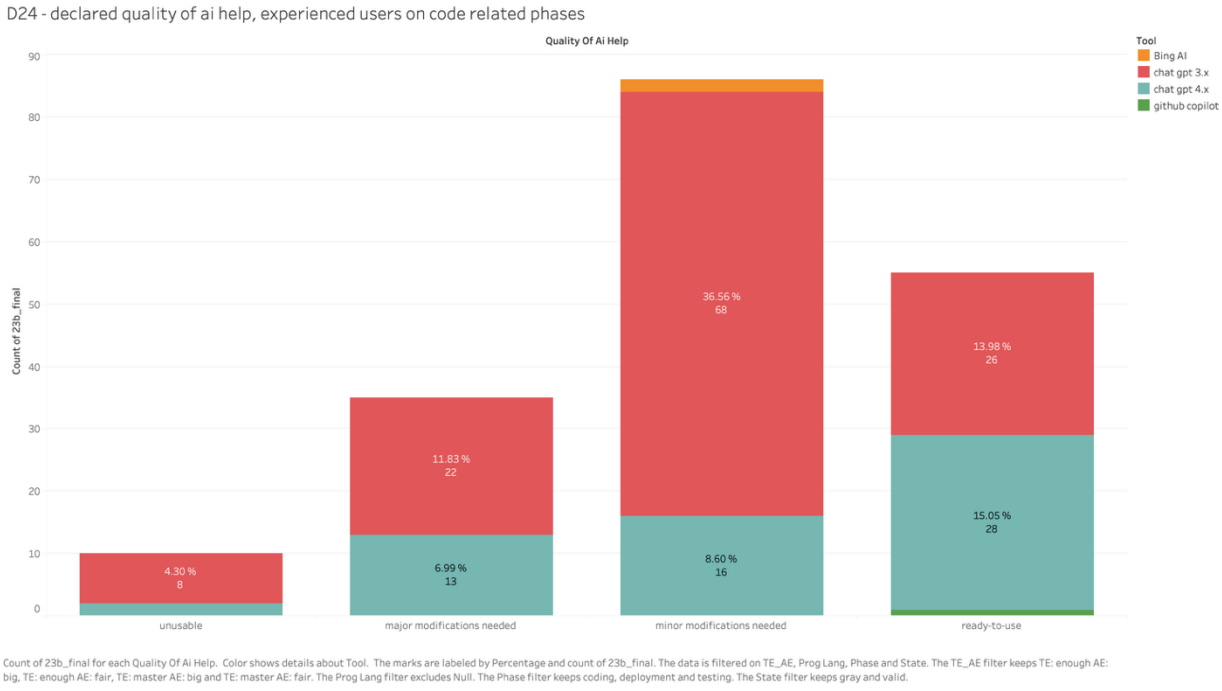

Figure 37.7.5 Declared quality of AI help by programming language – experienced users (softeng23b)

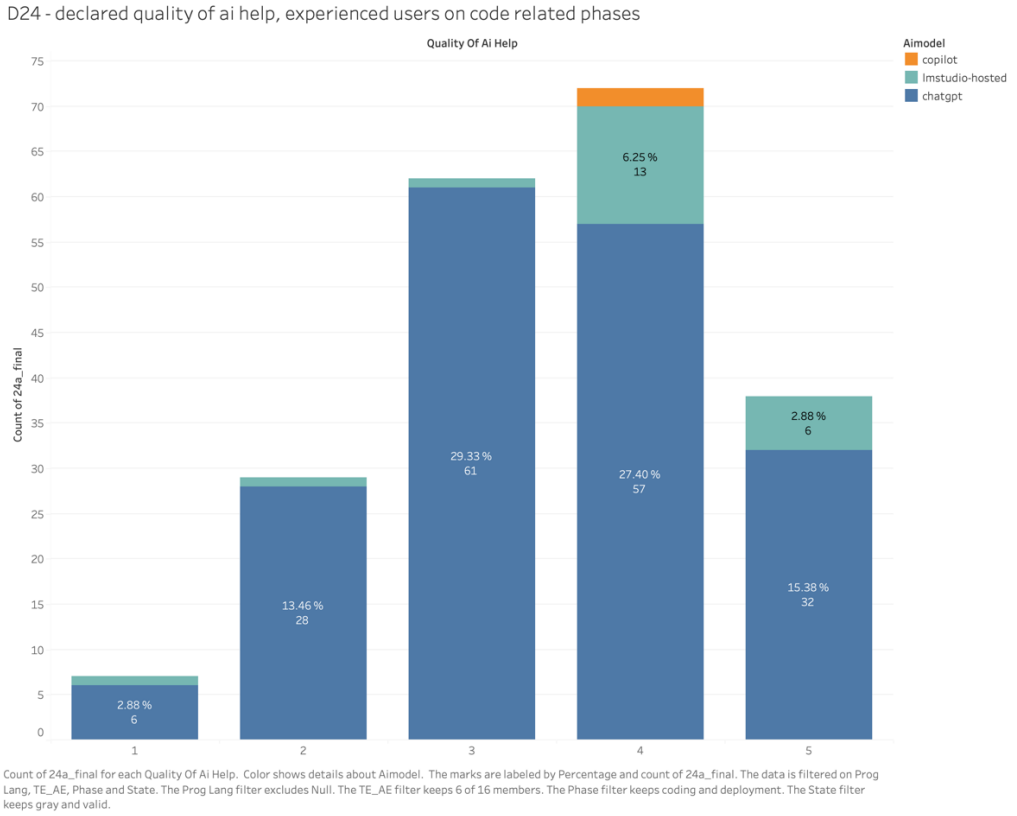

Figure 38.7.5 Declared quality of AI help by programming language – experienced users (saas24a)

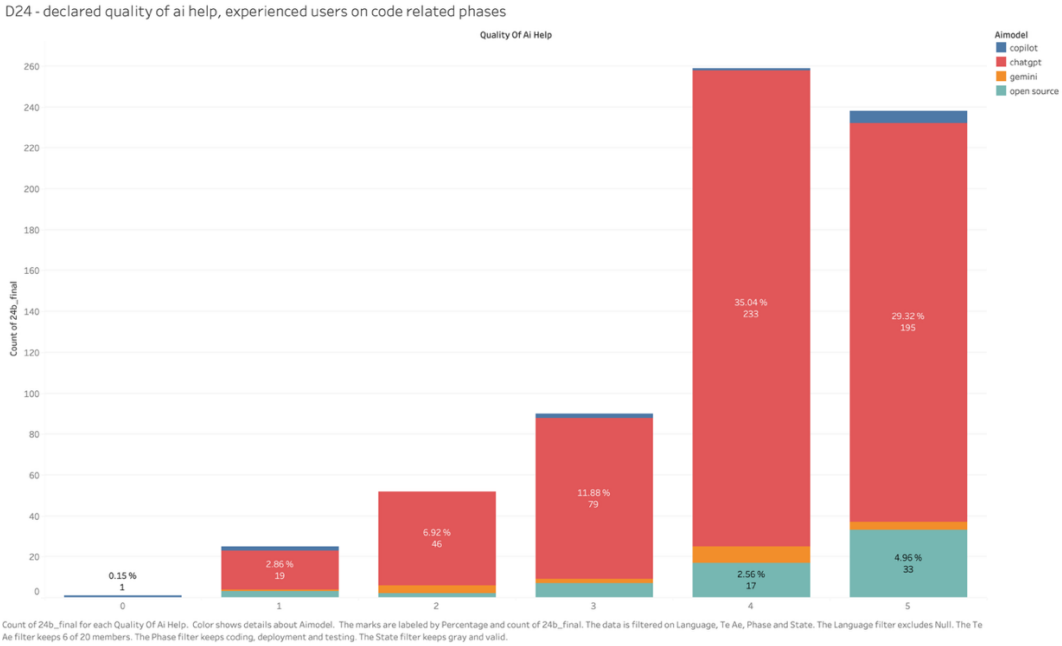

Figure 39.7.5 Declared quality of AI help by programming language – experienced users (softeng24b)

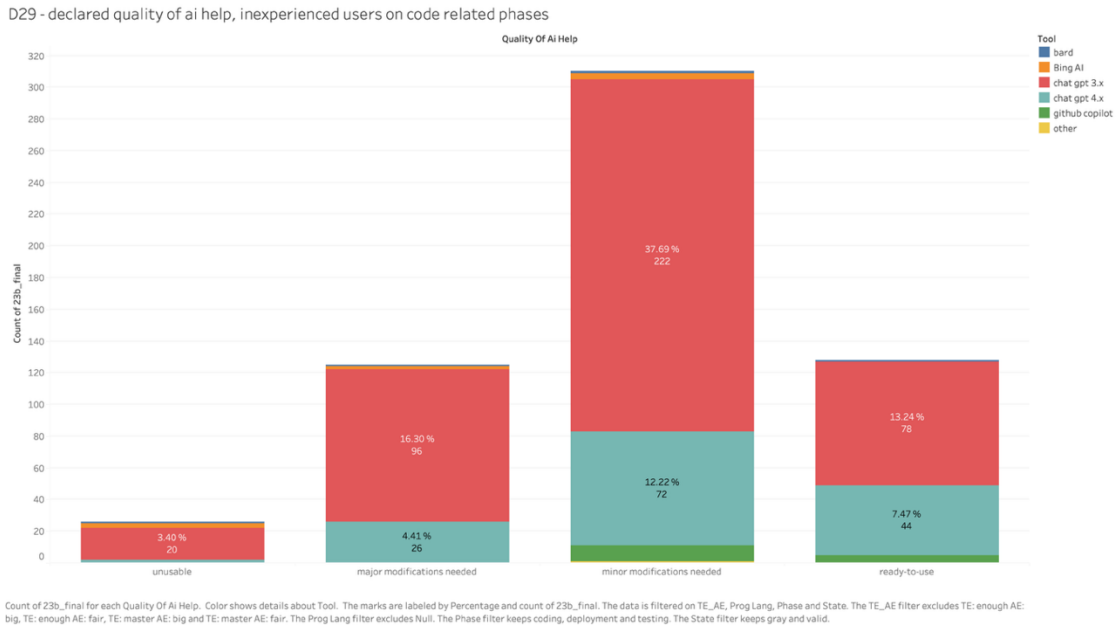

Figure 40.7.5 Declared quality of AI help by programming language – inexperienced users (softeng23b)

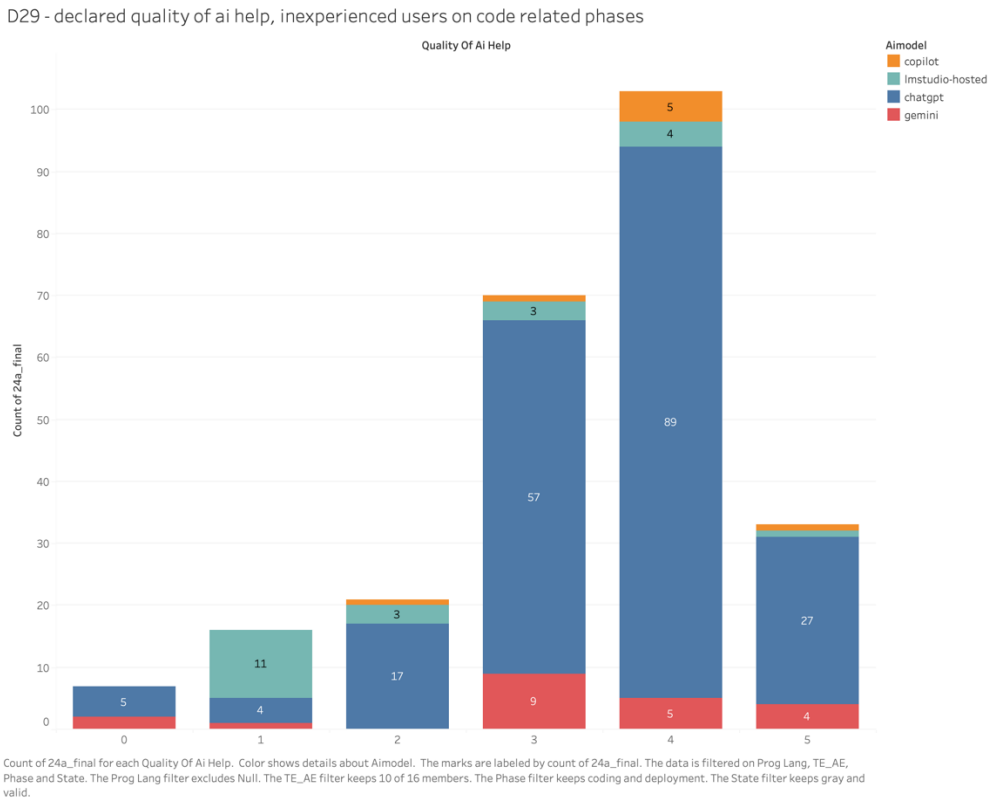

Figure 41.7.5 Declared quality of AI help by programming language – inexperienced users (saas24a)

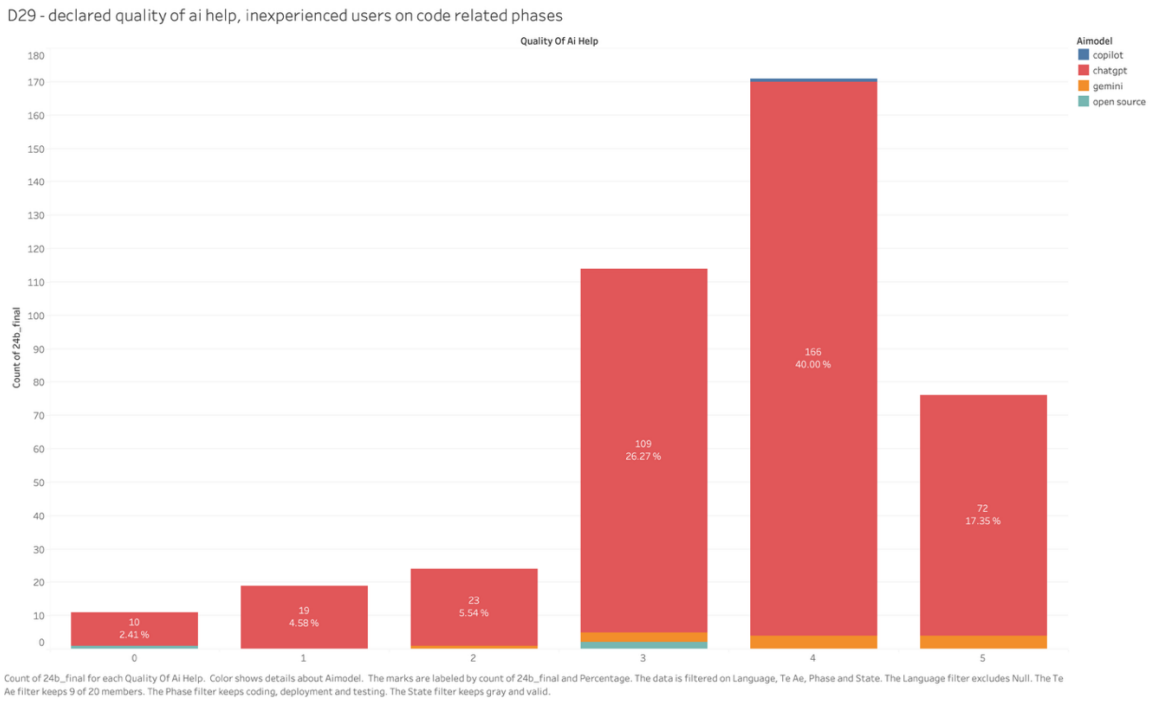

Figure 42.7.5 Declared quality of AI help by programming language – inexperienced users (softeng24b)

9. Answers to the Research Questions

9.1 RQ0: Can student-submitted data be reliably used for analysis?

Yes. The dataset was filtered using a structured validation matrix that checked each combination between phase-action-scope-tool. Invalid submissions (~17.6%) were removed for the analysis, and gray entries (~17.2%) along with the valid submissions (~65.2%) were kept for analysis. Coding phases across all datasets report the best validity rating at (>75%), while pre-code related phases had the lowest due to students' unfamiliarity. This filtering allows a high-integrity dataset to be suitable for the analysis.

9.2 RQ1: How do students use AI in non-code related phases?

AI is moderately helpful in pre-code phases such as requirements, architecture and design. Students mostly use AI as a brainstorming assistant, or a structural template provided in these phases. Quality ratings for AI outputs in these phases are concentrated around moderate scores with a slight skew to the right (3-4), with design receiving the highest evaluations and requirements gathering the most scattered. Time savings on the other hand are double or triple for the requirement phases, indicating that AI's utility even when output quality is imperfect still helps significantly. Inexperienced students show minimal threat perception, while experienced ones are more aware of AI's limitations and do not consider these phases as a threat.

9.3 RQ2: How do students use AI in code related phases?

AI quality appears to be the strongest in code-related phases, especially coding and testing. Students consistently rate AI-generated code as "ready-to-use" or with "minor modifications needed" (4-5 on the scale from 0 to 5). Time saved to time allocated ratio is close to double for each phase. Experienced users show moderate threat perception, acknowledging AI's usefulness and risk of replacement. Inexperienced users report even lower threat level, showing how unaware they are about AI replacement.

9.4 RQ3: Does programming language affect AI-perceived usefulness?

Yes. Python and JavaScript are the most used and highest-rated languages in terms of AI quality output. Submissions involving these languages are frequently rated 4-5 in perceived quality, especially by experienced users who are able to evaluate and modify accordingly. Languages like SQL or YAML have far fewer submissions but still follow the same distribution format with 4 having the highest count.

9.5 RQ4: How does student experience influence perception and effectiveness of AI?

Experience level significantly affects how students perceive and use AI tools. Experienced users are more selective, that AI output more critically and are more aware of potential threat. They use AI as a strategic assistant rather than using it entirely without checking what's being provided. Inexperienced users, tend to rate AI outputs highly, report minimal threat, often accepting outputs without critical revision. The difference is most visible in the code-related phases.

9.6 RQ5: How does AI perception evolve over time in tracked users?

Among students who were tracked in both courses (SoftEng-23b and SaaS-24a), AI perception improves remarkably. Tool and action experience scores increase across the board, emotional stance shifts from neutral to confident, and inexperienced users report more knowledge acquired. Quality ratings also improve, and there is a big increase in generic feeling for the future in both experienced and inexperienced users.

10. Limitations

While this thesis presents a detailed analysis of AI tool usage in software engineering education, several limitations must be acknowledged to contextualize the findings and guide future research.

First, the study does not include ground-truth validation of AI-generated outputs. Student-reported perceptions of quality and time savings were not verified against actual correctness, code performance, or alignment with project requirements. As such, conclusions about AI effectiveness rely on subjective metrics rather than objective benchmarks.

Second, the dataset lacks any assessment of prompt quality. Since the structure and formulation of prompts directly affect AI output, variations in student prompting skill may influence perceived utility. However, no standardized evaluation of prompts was included, and optional prompt logs were excluded from the main analysis. This omission limits the ability to distinguish between model limitations and suboptimal user input.

Third, the study population is drawn exclusively from advanced undergraduate students at the National Technical University of Athens. These students share a relatively homogeneous academic background, institutional training, and course structure. As a result, the generalizability of findings only makes sense in similar contexts. The results may not extend to students in non-engineering disciplines, to developers in professional environments, or to self-taught programmers with different tool exposure and project dynamics.

11. Future work

One meaningful next step would be to collect and analyze data from the upcoming SaaS-25a course. This would allow for a deeper comparison with SoftEng-24b, especially for students who enroll in both. Just as this thesis already tracked students across SoftEng-23b and SaaS-24a, adding this new group would help show how students' use of AI tools changes as they continue through different courses. It would also give a better picture of how earlier experience affects later choices, confidence and experience.

Another important improvement is the way data is collected. Right now, students have to prepare and upload files manually, which adds extra steps and can feel tiring—especially when repeated often. In future versions, it would make sense to use a simpler system. For example, a web platform could let students open AI chat windows directly—either by using their own API key or by accessing models hosted by the course team. This would allow interaction data to be recorded more smoothly, without interrupting their workflow. It would also help reduce errors and make it easier to collect high-quality data without adding extra effort for students.

12. Conclusion

This thesis has shown that AI tools are no longer experimental in a software development cycle, but they are fully integrated into the development workflow—particularly in code related phases—where students report consistent quality, very strong time efficiency and somewhat concern about AI replacement. AI is not merely saving time but it is reshaping how students approach problem-solving, brainstorming, unknown terminology definition and code organization. Even students with limited experience find AI to be a tool that helps get the ball rolling in the early phases, while the more advanced students use AI more strategically.

Before drawing any conclusions from the data, it was necessary to ensure that the submissions used actually represented a meaningful set of AI use. Each submission was carefully validated to check whether students had accurately documented what they were doing or if they were submitting random nonsense. Coding tasks were the most consistently well-logged cause students clearly understood how to describe their AI use and used a correct combination set of phase-action-scope. Students had more misclassification rates in the earlier phases, that could be due to them being unaware of each terminology and definition and difficulty in the submission of their AI interraction. The invalid and duplicated entries were marked as invalid.

In earlier stages—pre-code related phases like requirements, architecture or design—AI is less precise but still provides a moderately good quality for help. Here students treated the tools used as less of a solution and more as an assist on how they should start the development. These interactions still provide data for the time savings, even when the AI's output was declared as unusable. The role of AI in these abstract phases is less about correctness and more about the initiation of the project.

The majority of submissions was for the phases of coding and testing. Students focused heavily on AI tools to assist with implementation tasks, code management and source code authoring. Coding in particular, emerged as the most productive phase, since students received most often output that was ready-to-use, especially in programming languages like Python and JavaScript. The testing phase was reported to have slightly less quality rating overall, but higher time savings—suggesting that AI didn't only help by generating code in this phase but also clarified and explained logic for test structures and edge cases. Overall, across all code-related phases students reported having almost twice the time saved from the time they allocated in—which is safe to say that AI meaningfully sped up the development of the project. Experienced users saw AI as helpful but reported more on the threat level than the inexperienced users did, this shows that the perception is shaped by

each student's understanding, the more students grasp the task, the more they realize what's happening, and how much of a threat of replacement AI is.

The dataset that was used among the students who participated in both courses—SoftEng-23b and SaaS-24a—confirms that the repeated use helped the students become more confident and selective. In the first course, many rated themselves as only moderately familiar with the AI tools they used but by the second course, nearly all described themselves as experienced in the AI use. Interestingly, both experienced and initially inexperienced users reported similar emotional stances towards AI by the end. Inexperienced students showed the most growth in knowledge acquired since they focused on AI tools as a learning tool. Experienced users used the AI tools with more caution but still registered gains in trust—lower threat level—and learning. Across both groups, what emerged was the evolution of students, they learned where AI helps more and is more efficient and also how to use it strategically.

In the end, this study shows that AI tools have become deeply integrated into how students build software engineering projects. Not just for coding, but across the entire developmental cycle. Whether they use it as terminology definition, debugging aid, or a source for time saver—AI shaped how students worked and how they learned from it. Their judgments about quality usefulness and threat weren't fixed, they changed as students gained experience. Students are already living in the use of AI in software development, therefore, if given structured opportunities to engage with it they are already ready.

13. Appendix A: Valid paths for phase-action-scope

13.1 Paths for softeng23b

13.1.1 Filter for requirements gathering

```
{
    "valid":{
        "phase": ["requirements gathering"],
        "action": ["problem understanding", "stakeholder statement",
"requirements (functional)", "requirements (non-functional)", "use case
specification"],
        "scope": ["documentation (text)", "uml activity", "uml sequence"],
        "tool": ["chat gpt 3.x", "chat gpt 4.x", "bard", "other", "<fill
in>", "Bing AI"]
    },
    "gray": {
    }
}
```

13.1.2 Filter for requirements specification

```
{
    "valid":{
    "phase": [ "requirements specification"],
    "action": ["problem understanding", "requirements (functional)",
"requirements (non-functional)", "use case specification"],
    "scope": ["documentation (text)","uml activity", "uml sequence"],
    "prog lang": ["null", "other"],
    "other prog lang": ["<fill in>"],
    "tool": ["chat gpt 3.x", "chat gpt 4.x", "bard", "other", "Bing AI"]
    },
    "gray": {
    }
}
```

13.1.3 Filter for architecture

```
{
    "valid":{
    "phase": ["architecture"],
    "action": ["architectural decision"],
    "scope": ["uml component", "uml deployment", "uml class", "uml other",
"database design", "frontend", "data management", "backend", "api",
"deployment scripts"],
    "prog lang": ["null", "other"],
    "other prog lang": ["<fill in>"],
    "tool": ["chat gpt 3.x", "chat gpt 4.x", "bard", "other", "Bing AI"]
    },
    "gray": {
        "action": ["problem understanding"],
        "scope": ["frontend", "cli"]
    }
}
```

13.1.4 Filters for design

```
{
    "valid":{
        "phase": ["design"],
        "action": ["design decision"],
        "scope": ["uml activity", "uml sequence", "uml component", "uml
deployment", "uml class", "uml other", "frontend", "backend", "api" ],
        "prog lang": ["null", "other"],
        "tool": ["chat gpt 3.x", "chat gpt 4.x", "bard", "other", "Bing AI"]
    },
    "gray": {
        "action": ["problem understanding"],
        "scope": ["cli"]
    }
}

{
    "valid":{
        "phase": ["design"],
        "action": ["data design"],
        "scope": ["uml class", "database design"],
        "prog lang": ["null", "other"],
        "tool": ["chat gpt 3.x", "chat gpt 4.x", "bard", "other", "Bing AI"]
    },
    "gray": {
        "action": ["problem understanding"],
        "scope": ["data management"]

    }
}
```

13.1.5 Filter for coding

```
{
    "valid":{
        "phase": ["coding"],
        "action": ["source code authoring", "code management"],
        "scope": ["frontend", "data management", "backend", "api", "cli",
"test execution scripts", "deployment scripts", "code management actions"],
        "prog lang": ["js", "js-node", "python", "sql", "nosql db", "html",
"other"]
    },
    "gray": {
        "action": ["problem understanding", "unit testing", "functional
testing", "integration testing", "performance testing","other testing", "dev-
ops"],
        "scope": ["test cases", "test code driver"]
    }
}
```

13.1.6 Filter for testing

```
{
    "valid": {
        "phase": ["testing"],
        "action": ["unit testing", "functional testing", "integration
testing", "performance testing", "other testing"],
        "scope": ["frontend", "backend", "api", "cli", "test cases", "test
code driver", "test execution scripts"],
        "prog lang": [ "js", "js-node", "python","html", "other"]
    },
    "gray": {
        "action": ["problem understanding"]
    }
}
```

13.1.7 Filter for deployment

```
{
    "valid":{
    "phase": ["deployment"],
    "action": ["design decision", "source code authoring", "dev-ops", "vm
operations", "container operations", "network operations"],
    "scope": ["frontend", "backend", "api", "deployment scripts"]
},
"gray": {
    "action": ["problem understanding"]
}
}
```

13.2 Paths for saas24a

13.2.1 Filter for architecture

```
{
    "valid": {
        "phase": ["architecture"],
        "action": ["orchestration design","microservices definition",
"container structuring"],
        "scope": ["uml sequence", "uml component", "uml deployment", "uml
other", "backend", "api"]
    },
    "gray": {
        "action": ["api design", "data design"],
        "scope": ["frontend"]
    }
}
```

13.2.2 Filter for design

```
{
    "valid": {
        "phase": ["design"],
        "action": ["api design", "orchestration design", "choreography
design", "data design"],
        "scope": ["uml sequence", "uml component",  "uml other", "data
management", "frontend", "backend", "api", "messaging design"]
    },
    "gray": {
        "action": ["container structuring","source code authoring"],
        "scope": ["uml deployment"]
    }
}
```

13.2.3 Filter for coding

```
{
    "valid": {
        "phase": ["coding"],
        "action": ["source code authoring", "network operations", "code
management"],
        "scope": ["frontend", "backend", "api", "deployment scripts", "github
operations"],
        "prog lang": ["js", "js-node", "python", "sql", "nosql", "java",
"yaml/json", "other"]
    },
    "gray": {
        "action": ["api design"],
        "scope": ["container configuration"]

    }
}
```

13.2.4 Filter for deployment

```
{
    "valid": {
        "phase": ["deployment"],
        "action": ["container structuring", "source code authoring", "network
operations"],
        "scope": ["uml deployment", "frontend", "backend", "api", "messaging
deployment", "container configuration", "deployment scripts", "github
operations"],
        "prog lang": ["n/a", "js", "js-node", "python", "java", "yaml/json",
"other"]
        },
    "gray": {
        "action": ["code management"]
    }
}
```

13.3 Paths for softeng24b

13.3.1 Filter for requirements gathering

```
{
    "valid":{
        "phase": ["requirements gathering"],
        "action": ["problem understanding", "stakeholder statement",
"functional requirements", "non-functional requirements", "use case
specification"],
        "scope": ["uml activity", "uml sequence", "uml other"]
    },
    "gray": {
    }
}
```

13.3.2 Filter for requirements specification

```
{
    "valid":{
        "phase": ["requirements specification"],
        "action": ["problem understanding","functional requirements",
"non-functional requirements", "use case specification"],
        "scope": ["uml state", "uml activity", "uml sequence", "uml
other"]
    },
    "gray": {
        "scope": ["data management", "backend", "api"]
    }
}
```

13.3.3 Filter for architecture

```
{
    "valid":{
        "phase": ["architecture"],
        "action": ["architectural decision"],
        "scope": ["uml component", "uml deployment", "uml other", "data
management", "backend", "api"]
    },
    "gray": {
        "action": ["problem understanding", "design decision"],
        "scope": [ "frontend" ]
    }
}
```

13.3.4 Filter for design

```
{
    "valid":{
        "phase": ["design"],
        "action": ["design decision", "data design"],
        "scope": ["uml state", "uml activity", "uml sequence", "uml
component", "uml deployment", "uml other", "frontend", "backend", "api"]
    },
    "gray": {
        "action": ["problem understanding","functional requirements", "use
case specification","source code authoring","architectural decision"],
        "scope": ["data management"]
    }
}
```

13.3.5 Filter for coding

```
{
    "valid":{
        "phase": ["coding"],
        "action": ["source code authoring","code management"],
        "scope": ["data management", "frontend", "backend", "api",
"messaging design", "container configuration", "github operations"]
    },
    "gray": {
        "action": ["problem understanding", "design decision", "data
design" , "unit testing", "functional testing", "integration testing",
"performance testing","other testing", "dev-ops"]
    }
}
```

13.3.6 Filter for testing

```
{
    "valid":{
        "phase": ["testing"],
        "action": ["unit testing", "functional testing", "integration
testing", "performance testing","other testing"],
        "scope": ["frontend", "backend", "api", "messaging deployment",
"deployment scripts"]
    },
    "gray": {
        "action": ["problem understanding"]
    }
}
```

13.3.7 Filter for deployment

```
{
    "valid":{
        "phase": ["deployment"],
        "action": ["design decision", "source code authoring", "dev-ops",
"vm configuration", "container configuration","network configuration"],
        "scope": ["frontend", "backend", "api", "deployment scripts"]
    },
    "gray": {
        "action": ["problem understanding"]
    }
}
```

14. Appendix B: A sample .json entry

14.1 softeng23b random submission


```
{
    "answers": {
        "phase": "requirements gathering",
        "action": "requirements (non-functional)",
        "scope": "documentation (text)",
        "action experience": "fair",
        "prog lang": "n/a",
        "other prog lang": "<fill in>",
        "tool": "chat gpt 4.x",
        "other tool": "<fill in>",
        "tool option": "full",
        "tool experience": "master",
        "time allocated (h)": "0.25",
        "time saved estimate (h)": "2",
        "quality of ai help": "major modifications needed",
        "generic feeling": "great as-is",
        "notes": "<fill in>"
    }
}
```


14.2 saas24a random submission


```
{
    "answers": {
        "phase": "architecture",
        "action": "network operations",
        "scope": "backend",
        "action experience": 4,
        "prog lang": "js-node",
        "other prog lang": "None",
        "aimodel": "chatgpt",
        "aimodel version": "4.0",
        "lmstudio-hosted aimodel": "No",
        "tool option": "online full",
        "experience with tool": 5,
        "time allocated (h)": "2",
        "time saved estimate (h)": "3",
        "quality of ai help": 4,
        "knowledge acquired": 3,
        "generic feeling - now": 3,
        "generic feeling - future": 4,
        "threat level": 5,
        "notes": "None"
    }
}
```

15. Bibliography

[1] J. e. a. Prather, Novice Programmers' Patterns with Copilot, https://arxiv.org/abs/2304.02491, Ed., arXiv, 2023.

[2] R. a. C. P. Alves, "Give me the code": First-Year Student ChatGPT Logs, https://arxiv.org/abs/2411.17855, Ed., arXiv, 2024.

[3] E. e. a. Shihab, Copilot in Brownfield Programming Tasks, https://arxiv.org/abs/2506.10051, Ed., arXiv, 2025.

[4] M. e. a. Kazemitabaar, Youth Learners Using LLM Code Generators, https://arxiv.org/abs/2309.14049, Ed., arXiv, 2023.

[5] A. a. A. D. Elbanna, Exploring the Integration of ChatGPT in Education, https://doi.org/10.1108/msar-03-2023-0016, Ed., Emerald Publishing, 2023.

[6] A. P. R. a. C. L. Haldar, Assessing the Effectiveness of ChatGPT in Preparatory Testing Activities, https://doi.org/10.1109/fie61694.2024.10893214, Ed., IEEE, 2023.

[7] M. a. L. X. Heathen, A Review on the Perks of Using ChatGPT in Education, https://doi.org/10.20944/preprints202406.1060.v1, Ed., Elsevier, 2023.

[8] Y. Z. H. W. M. a. C. L. Liu, Response Strategies to ChatGPT in Language Education, https://doi.org/10.38007/ijeis.2023.040114, Ed., IJEIS, 2023.

[9] T. U. S. a. A. F. Rasul, The Role of ChatGPT in Higher Education, https://doi.org/10.37074/jalt.2023.6.1.29, Ed., JALT, 2024.

[10] C. H. Q. a. W. Y. Sun, ChatGPT-Facilitated Programming Tasks, https://doi.org/10.1186/s41239-024-00446-5, Ed., ACM, 2023.

[11] C. H. Q. a. W. Y. Sun, Prompt-Based Learning with ChatGPT, https://doi.org/10.1057/s41599-024-03991-6, Ed., ACM, 2023.

[12] X. G. S. a. L. J. Ma, Eight-Week Use of ChatGPT in Python Course, https://arxiv.org/abs/2403.12030, Ed., arXiv, 2023.

[13] T. Z. Y. a. D. L. Wang, Role of AI Literacy in Using ChatGPT for Code, https://doi.org/10.1145/3624062.3624225, Ed., IEEE, 2024.

[14] A. C. L. a. P. R. Haldar, AI in Preparatory Software Testing, https://doi.org/10.1109/fie61694.2024.10893214, Ed., IEEE, 2024.

[15] A. Z. Y. a. Q. J. Jalil, ChatGPT on Software Testing Course Questions, https://doi.org/10.48550/arXiv.2302.03287, Ed., arXiv, 2023.

[16] M. Qureshi, ChatGPT in Undergraduate Data Structures Course, https://arxiv.org/abs/2308.10797, Ed., arXiv, 2023.