The Impact of Generative Artificial Intelligence Tools in Project-Based Learning

Tom van Dijk[0000−0002−5366−1051] and Vadim Zaytsev[0000−0001−7764−4224]

Formal Methods and Tools
University of Twente, Enschede, The Netherlands
t.vandijk@utwente.nl, vadim@grammarware.net

Abstract. We investigate the potential risks and benefits for students utilising Generative Artificial Intelligence (GAI), specifically OpenAI's ChatGPT and GitHub's Copilot, to generate solutions instead of independently creating them in the traditional educational setting. The rapid advancements in GAI have transformed numerous domains, including software development and software engineering education. While these tools offer unprecedented convenience and efficiency, there are growing concerns regarding their potential implications for academic integrity and genuine student learning. We report on a pilot study in which 40 students, who completed a first-semester course on object-oriented programming, re-engage in a comparable programming project in limited time using GAI tools. In this pilot study, we aim to assess the extent to which students can rely on GAI tools to generate solutions for large programming assignments, to investigate the impact of AI-driven code generation on students' understanding of fundamental programming concepts and problem-solving abilities, and to explore the perspectives of educators and students on the implications and long-term consequences of integrating AI-assisted coding tools in the learning process.

1 Motivation and Background

Generative Artificial Intelligence (GAI) is a field of study that uses trained models to generate data fragments shaped as coherent text, realistic images, believable videos and executable code. These models can be generative adversarial networks [17], variational autoencoders [22], diffusion models [9], autoregressive models [20], energy-based models [41], flow-based models [35], as well as — the most famous these days — large language models. Recent rapid advancements in this area have made a lot of things possible: instead of requiring large volumes of specific data, modern GAI tools can combine large volumes of generic data they have already witnessed, with a concise definition of context, to generate all kinds of artefacts traditionally required as deliverables in the software industry as well as software engineering education: software, documentation, solutions for given problems, tests for given code, presentations, critical reflections, etc. This in turn started reshaping various professional fields from creative industries

to software development. In the literature this phenomenon is known as *job crafting* [14], because it leads to employees reshaping their jobs to fit evolving demands. Tools such as ChatGPT [7] and GitHub Copilot [16] are now capable of generating entire blocks of text or code with minimal human intervention, potentially accelerating productivity in a significant way.

However, in education — in particular in programming courses — this technological leap presents not just opportunities, but mostly challenges. While GAI-powered tools can assist students in writing code, testing, debugging, documenting, even learning new concepts, they also raise fundamental questions about academic integrity, skill acquisition, and the role of traditional learning methods. Educators can no longer afford to ignore GAI; instead, we must critically examine its impact and develop strategies to either integrate or regulate its use in ways that enhance, rather than diminish, student learning. In this study, we aim to explore how students interact with GAI in a project-based programming course, assessing whether these tools support or hinder their understanding of fundamental programming concepts.

There is an actively ongoing heated debate among educators and researchers in education concerning the impact of GAI, whether it constitutes a threat or an opportunity [4, 5, 40, 48, 49]. On one side, critics argue that GAI can undermine academic integrity, diminish critical thinking skills, create new forms of inequality through differential access to advanced technological tools, etc. On the other side, proponents highlight the capacity of GAI-based systems to personalise learning experiences beyond what is possible by human means, streamline administrative tasks to redirect human teachers' attention to the most impactful places, and promote more inclusive pedagogical strategies by fully accommodating diverse learning styles. One more complementary viewpoint is that the use of GAI tools is not inherently bad, and potential productivity boosts it delivers are attractive, but with one unavoidable precondition: the user must understand the output of such GAI tools. If this prerequisite is not fulfilled, then the quality of the result cannot be assessed and thus cannot be guaranteed. All these conflicting perspectives revolve around concerns of bias, transparency, ethical accountability, making it increasingly complex for educators, policy makers and researchers to take a definitive stance on the overall impact of GAI. Ultimately, the question of whether GAI is a threat or an opportunity remains open-ended, with the answer depending on a long list of global aspects like its adoption in the field, as well as seemingly small details like concrete studied topics, formulated learning goals and attitude of both learners and teachers.

ChatGPT [7] is a large language model developed by OpenAI. It is capable of generating text based on user prompts, and does it in a human-like fashion. In computer science education, it can function as a virtual tutor, assisting students with code explanations, debugging help, algorithm suggestions, as well as full program generation. Unlike traditional search engines, ChatGPT provides conversational, context-aware responses, making it an attractive tool for learning — if not for teachers, then definitely tempting for students. However, its ability to generate plausible and believable yet incorrect answers poses a challenge: without

fully understanding the underlying concepts, students may rely on it too much, potentially weakening their problem-solving skills. This duality — being both a powerful learning aid and a potential shortcut that bypasses learning deeper understanding — makes ChatGPT a critical subject of study in programming education.

Copilot [16], developed by GitHub in collaboration with OpenAI, is a GAI-powered code completion tool designed to assist programmers by suggesting entire functions, generating boilerplate code, and providing inline coding assistance. Integrated directly into coding environments like Visual Studio, VS Code, IntelliJ IDEA, Xcode and the like, Copilot offers real-time code predictions based on the developer's comments and existing code structure. For students, Copilot can accelerate coding tasks, reduce syntax errors, and serve as an example-driven learning tool. However, similar to ChatGPT, it raises concerns regarding over-reliance and passive learning, as students may accept AI-generated solutions without fully understanding the logic behind them. The industry's reaction to GitHub Copilot has been mostly positive, but unlike expert developers who use it as a powerful code completion tool which output they can fully comprehend before committing to it, novice developers like students still learning how to program properly, can just end up using it as a shortcut to a *somehow* working solution, bypassing the actual learning.

1.1 Problem Statement

In this paper, as well as in our programme, we rely on the paradigm of *project-based learning* [15,23]. It is an educational approach where students are supposed to gain knowledge and skills by actively engaging in real-world, complex projects over an extended period. We see this as a programme-level implementation of the university-wide vision of *learning by interacting* [47], but it is also closely related to concepts of *problem-based learning* [33,52], which focuses on solving open-ended problems while defining your own learning outcomes; *challenge-based learning* [46,51], which emphasises real-world challenges which are often interdisciplinary and by definition larger than what learners can solve, which forces them to choose their own challenges and face them; as well as *student-driven learning* [12,21], which is an approach that provides personalised learning experiences by moving away from traditional teacher-directed models. With some small variations (which might be very important on implementation level), all these approaches share the line of thinking that values principles of critical thinking, problem solving, making own decisions on the learning path, influencing goals, outcomes, deliverables and strategies.

Traditionally, project-based learning ensures deep engagement by requiring students to design, implement, refine and improve their own solutions. However, the integration and/or mere existence of GAI tools challenges this paradigm. The students who employ these tools, gain the ability to generate significant portions of their projects with minimal effort, potentially diminishing the learning-by-doing aspect. While GAI can serve as a tutor or assistant in overcoming technical hurdles, its automation of cognitive processes risks turning project work into a

passive exercise of prompt engineering *instead* of genuine problem solving. In this study we aim to contribute to investigation of how GAI affects traditional project-based learning setups, determining whether it supports learning or unintentionally undermines the development of essential programming competencies.

Since project-based learning thrives on the principle of learning-by-doing, where students engage directly with coding challenges, wrestle with design decisions, debug their own errors, refine their solutions, and learn from engaging in all those activities with their own freshly made artefacts, using GAI to substitute or "optimise" some of those steps naturally raises a fundamental concern. If GAI performs the "doing", does any meaningful *learning* still take place at all? When students shift from actively constructing solutions to passively prompting a GAI tool for answers, they may miss out on the deep engagement required to develop core programming competencies. This shift threatens skill acquisition [18,54], as students risk developing fluency in prompt engineering rather than algorithmic thinking, debugging strategies, or software design principles. Beyond the pedagogical concerns, this raises pressing academic integrity issues [39]. If a GAI tool generates substantial portions of a project, how do we assess a student's true understanding of the material? The challenge for educators is to differentiate between GAI-assisted learning and GAI-driven substitution, ensuring that students still engage in the cognitive processes that build genuine programming expertise. Traditional plagiarism detection methods are often ineffective against GAI-generated content [50]. Addressing these concerns requires new assessment strategies, clear ethical guidelines, and a rethinking of how learning is measured in a GAI-assisted educational landscape.

Lastly, the existence and commoditisation of GAI tools may or may not enforce a fundamental reevaluation of how *learning outcomes* are defined and assessed in programming education. Traditionally, assessments focus on code quality, problem-solving ability, conceptual understanding, often measured through project work, exams, or written reports. However, when GAI can generate functional deliverables, including code, documentation, and even debugging explanations, educators must ask: are our current learning outcomes still meaningful, or do they need to evolve? One viewpoint could be to maintain existing learning goals but adjust assessment methods — for example, shifting toward oral exams, in-class coding tasks, or GAI-aware rubrics that evaluate reasoning rather than output. Another, more radical approach could be to overhaul learning objectives entirely, recognising that in a GAI-assisted world, the emphasis should shift from syntax and implementation toward embracing GAI collaboration, debugging GAI-generated solutions, and verifying correctness. Whether through adaptation or reinvention, assessment must ensure that students develop not just the ability to produce code but also the critical thinking skills necessary to navigate a world where GAI is an inevitable part of the software development process. We leave this aspect as a dilemma for future work.

In this pilot study, our goals are:

- ◇ to assess the extent to which students can rely on GAI tools to generate solutions for programming assignments;

- ◇ to explore the perspectives of educators and students on the implications and long-term consequences of integrating AI-assisted coding tools in the learning process.

We hope that these preliminary collected results will help in the near future to make the next step to investigate the impact of AI-driven code generation on students' understanding of fundamental programming concepts and problem-solving abilities.

1.2 Context

In the Bachelor programme of *Technical Computer Science* at the University of Twente, object-oriented programming and software design are taught as an introductory course in the second quartile of the first study year, in a period of 10 working weeks.

The course consists of a 4 EC part where students learn about software modelling with UML diagrams about the development process in general, and an 8 EC part where students learn programming using Java, study basics of concurrency and race conditions, as well as basic networking with sockets. The students spend roughly 7 weeks practising to reach these learning goals, and then do a project in the remaining 3 weeks.

This project involves writing a full server-client application using Java Sockets. This application allows users to play a simple board game over the network against other players and against a simple computer player that is also programmed by the students. Typical board games for this project are perfect information games played on a rectangular grid, with few rules and simple game-over conditions, such as Connect-Four, Pentago, Othello and others. Students typically perform this project in pairs, although typically some students do the project by themselves when their partner drops out.

2 Related Work

There are many GAI tools provided by different vendors, but we focus specifically on GitHub Copilot [16] and ChatGPT [7], because they either perform competitively when matched against alternatives, or outperform them significantly. Both of them are also rather well-known and easy to setup, which in turn leads to them being well researched [30].

GAI repeats things it learnt from the training data, which also includes common mistakes [5], security vulnerabilities [36] and copyright violations [10]. This also leads to bias which some researchers see as hallucinations [48], while others equate hallucination to inventing believable terms [49], which is the opposite to repeating common mistakes. In classical terms, a GAI tool which gives an incorrect answer because it has witnessed it often before, can be seen as generating a *false negative*, while a GAI tool that produces an incorrect but seemingly convincing answer because it finds a way to concoct a wrong but believable chain of arguments leading to it, can be seen as *false positive*.

Interestingly, repeating commonly observed patterns does not mean GAI tools like Copilot will suggest idiomatic code, at least not for JavaScript and Python [38]. Presumably this happens because there is enough non-idiomatic code in existence, which is known from modernity research, investigating intricate patterns of coevolution of languages and codebases [1, 13]. Another issue potentially limiting Copilot's applicability in practice is its lack of robustness, meaning that a slight change in its prompt might make it generate a completely different completion code [30].

Future education is projected to include techniques like *intelligence augmentation* [40], which will emphasise skills like prompt engineering (not just knowing the syntax or patterns, but also conceptually "asking the right questions"), and *virtual intelligent tutoring*, when an interactive LLM like ChatGPT essentially replaces traditional teachers' tasks like providing additional explanations and formative feedback [40, 48].

Many authors point out that (over)reliance on GAI will train learners less in skills like critical thinking [48, 49]. Some also argue that this skill needs to be trained additionally next to similar activities like developing emotional intelligence [18]. On the positive side, it is exactly this lack of overly critical attitude towards its output that leads to their admitted usefulness in gathering ideas and issues on any given topic [49]. Unfortunately, specifically in software engineering overly creative out-of-the box solutions tend to contain more bugs than solutions following the beaten track.

It is becoming increasingly more common to see more technically literate students rely on GAI tools as a "clever" shortcut to completing homework effortlessly, and less GAI aware teachers condemning these actions and generalising them to any GAI use. Discussions of using GAI in the classroom commonly gravitate towards the topic of plagiarism [49]. Some authors go as far as firmly and inherently linking the use of ChatGPT and similar GAI tools to academic dishonesty [39]. However, even if there are some automated detection options available for the by now hopelessly outdated ChatGPT 3.5, most of them desperately fail to detect the use GPT 4, let alone o1, o3 or 4.5 [50].

When it comes to institutional policies, the most common one is to "ban it till we understand it" [27], or, to put it more mildly, to have "a comprehensive strategy to manage AI-assisted learning and control AI tool usage within educational institutions" [49]. Most such policies revolve around two principles: (1) banning GAI by default unless explicitly permitted by a deviant teacher; and (2) demanding explicit claims of using GAI or even not using it, possibly to create legal liability. This is very much in line with the early attitude and policies against using Wikipedia in classroom [19].

Yan et al list "AI-induced performance illusion" among the key challenges for learners [54]. This is, of course, the cornerstone of the entire issue: GAI tools are just tools which need to be learnt to be used effectively, and at their current state their output must be examined with not much less rigour than results of one's own manual coding or the code of one's junior colleague. GAI is good learning

aid and a decent completion instrument, but it has many downsides as a shortcut since it tempts users to bypass skill acquisition.

Specifically in the context of project-based learning, there are some recent BSc and MSc theses that attempted to enhance GAI tools' functionality of providing specific feedback to students while avoiding giving them solutions as such, and at the same time explicitly teaching students how to seek such feedback [25, 44]. The consensus at the moment seems to be that it is possible at least to some extent to restrict and/or customise GAI to provide feedback, and asking for it is definitely a teachable skill, but it remains to be seen whether such generated feedback truly contributes positively to the learning experience. In any case, it seems to decidedly improve peer feedback [43]. We see this line of work as complementary to ours.

Just like students have an option to use GAI for solving assignments given to them, teachers can use GAI to assess students' deliverables automatically. It is known that when it comes to essays and small study reports, GAI scores are comparable to those given by human examiners [26, 32], as long as complexity is kept low. For project-based learning, there are some emerging frameworks being proposed for GAI-driven automated grading [53], but as of now they lack substantial evidence that would demonstrate the limits of their applicability. However, from a line of research we pursue in parallel to this work, we know that not only automated grading is prohibited by many Examination Boards, it is also highly undesirable when stakeholders are consulted [42]. What stakeholders — from teachers to students and from e-learning experts to educational management — want, is automated *assistance* in their assessment activities, not automated grading per se. Such assistance, of course, can be, among many other methods, based on (G)AI.

Lastly, there are some recent examples of project-based courses specifically designed to integrate GAI models and tools into structured co-creation activities [28, 37, 45]. Since courses that do not undergo such redesign, tend to report that all students use GAI anyway [6], avoiding this issue seems like a losing battle.

3 Methodology

We have recruited students who passed the first year of the Bachelor programme in Technical Computer Science, that is, who completed the course in question before, including the project of implementing a board game. To level the playground and to make sure nobody of them can reuse their own code, we have asked them to implement a different board game.

We have divided the students up into four groups of 10 students each:

- ◇ Control group, not allowed to use any GAI tools at all
- ◇ "Copilot" group, instructed to use Copilot and not allowed to use ChatGPT
- ◇ "ChatGPT" group, instructed to use ChatGPT and not allowed to use Copilot
- ◇ "Both tools" group, instructed to use both Copilot and ChatGPT

To divide the students in groups, we first asked the students to self-assess their programming ability, their skill to use Copilot and their skill to use ChatGPT. We then placed the students into categories and randomly split the students from each category into the four groups.

The programming project requested from these students, was slightly more difficult than the official project of the previous year. They were instructed to implement the game Hex, which requires reasoning about hexagonal geometry (rather than playing on a grid of squares), and to implement pathfinding from one side of the board to the other. These requirements are usually deemed too difficult for students in the first semester, since on the starting day some students do not even know the basics of programming, but would be interesting challenges for students who have already completed their first year. Furthermore, we instructed students to spend in total exactly 36 hours on the programming project, which is less than the roughly two weeks in the first semester, even accounting for student working hours being shorter than those of professionals. If they were finished earlier, they could spend more time polishing their implementations to improve their grade. Students from groups with an allowed use of GAI, were instructed to do so as much as would be reasonable to get a high grade.

During the project, the participants maintained a *reflective journal*, ideally requiring about 5–10 minutes per two hours of their programming work. After completion of the project, they *self-assessed* their work using the official rubric, which is the same rubric of the project in the first-semester course. They then *peer reviewed* the grading of another project. Finally, the projects were discussed in *focus group meetings*.

The reflective journals, comprising 120980 words in total (after removing code snippets, screenshots and diagrams), were read and manually tagged with the following tags:

- Judgemental tags:
  - **joy**: any expression of positive emotion, satisfaction with the result, with GAI interaction, tool behaviour, etc, beyond "works as expected".
  - **pain**: any expression of negative emotion, frustration, dissatisfaction, dislike of results or process steps.
  - **boost**: any place in the text where students reported a significant help or speedup caused by the use of GAI.
  - **waste**: any place where the use of GAI was accompanied by a waste of time and resources, including dysfunctional code, misleading designs, bugfixes that did not result in desired effect, etc.
  - **manual**: admitting manual programming (usually after a failed attempt of GAI use or intentionally replacing it).
  - **prompt**: any place where a prompt was described or quoted, or a moment when GAI was "asked" or otherwise queried.
  - **could have**: any places where GAI was used but the result was perceived as something the user could have done themselves, or places where the user deliberately decided to not use GAI while having that option.
  - **would have**: any places where the user would have wanted to use GAI but did not because it was not permitted by the rules of their group.

- **ai misuse**: any description of situations where GAI was misused and/or could have been misused.

◇ Coding activity tags:
- **project planning**: taking a step back and contemplating next steps in the implementation.
- **refactoring**: cleaning up and otherwise improving the design of previously written or generated code.
- **debugging**: finding, localising, understanding or removing bugs from existing code.
- **testing**: checking existing code for certain properties or scenarios, writing test cases, executing them, augmenting the test suite, etc.
- **reasoning**: any activities related to understanding existing code, be it for the programmer's sake or for the sake of interpreting GAI's output or formulating a prompt.
- **documentation**: anything describing manipulations with comments, Javadoc [24], report or logbook.

◇ Design topic tags:
- **class diagram**: UML class diagrams, but only if explicitly mentioned (discussing object-oriented design did not lead to tagging).
- **sequence diagram**: UML sequence diagrams, but only if explicitly mentioned (discussing system behaviour did not lead to tagging).
- **design pattern**: usually MVC or Listener, but with some mentions of Factory.

◇ Specific topic tags:
- **game rules**: anything related to the game as such, most often some details like the form of the hexagonal board (and determining neighbours on such a board) or computing the winning condition.
- **protocol**: anything related to the mandated design of the communication between the client and the server, the kinds of commands that had to be supported, their order, etc.
- **concurrency**: anything related to the distributed nature of the system, usually about choosing a threadsafe data structure or suffering from consequences and race conditions when working with a non-threadsafe one.
- **network**: sending messages over the network, using sockets, occasionally figuring out details of client-server collaboration.
- **user interaction**: visualising the game board and communicating with the player about their moves.
- **error handling**: the entire spectrum from catching exceptions to just dealing with abnormal scenarios.
- **JML**: Java Modelling Language [8], mastering which was a part of the learning objectives of the course, and was meant as a way to invite students to reason about correctness of their code.
- **TDD**: test-driven development [31], which was encouraged in the original project (along other techniques like pair programming), and we expected to find it back in the reports, only to be gravely disappointed. TDD

was mentioned explicitly three times, and on one additional occasion a participant reported to write a deliberately failing test case before implementing the next feature, which is a TDD technique.

The tagging was done following the principles of narrative research [11] and should be taken at face value. For instance, if a student decided to explain their debugging process in five sentences, then each of them was tagged with the **debugging** tag, even when the issue was not five times larger than another debugging case. Similarly we dealt with emotion tags: for instance, if GAI generated dysfunctional code, simply reporting that it did not work would merit at most a **waste** tag, but writing down that it "forgot" or "misunderstood" the question or that it was not helpful, was read as opinions and marked with the **pain** tag.

Additionally, the first pair of tags (**joy** and **pain**) might seem similar to the second pair (**boost** and **waste**), and while it is true that dealing with dysfunctional code often triggers negative emotions, while getting visible performance gains usually goes the other way, this tendency was not universal. For example, the following sentences were tagged as both **pain** and **boost**:

- ◇ *I was not satisfied with this at all but gave at least the idea for the future code.*
- ◇ *Though it was quite tedious having to copy paste each class, one by one, it saved me a lot of time.*

4 Results

4.1 Quantitative Findings

We originally started with 40 participants. During the study, we lost 3 participants due to personal GAI-unrelated circumstances: two participants of the control group did not deliver anything and were excluded from the data, one more participant from the "both tools" group submitted a minimal journal which we ended up including in the data just to get a few more taggings (participant 4 on Table 3). Their journal was 393 words long, which is considerably lower than the average (mean 3184, median 2621), but not dramatically smaller than the shortest full journal (764 words).

After self-assessment and peer review, we summarise the grades that the projects would receive according to the rubric in Table 1. Unfortunately the sizes of the groups are such that results are **not** statistically significant. Still, we observe that groups with access to Copilot had better pass-fail outcomes, and that the group with access to both tools had both higher average and higher median grades. Due to the sizes of the groups, we cannot truly draw hard conclusions about the effect on the grades.

Daily/hourly journals which students submitted, provided much more detailed insights into their way of working and the impact of using GAI, which we will discuss in the next section. In total, these journals contained 120980 words to

			Average		
Group	**Size**	**Pass:Fail**	**Functionality**	**Quality**	**Report**
Control group	8	5:3	6.19	5.93	5.69
“Copilot” group	10	9:1	7.38	6.66	6.46
“ChatGPT” group	10	7:3	6.60	6.29	5.95
“Both tools” group	9	8:1	7.47	6.64	7.08

Table 1. Grades for the projects of each group, and for the Functionality, Software quality and Report subgrades. Grades are between 1 and 10 (excellent), where 5.5 is the minimum passing grade.

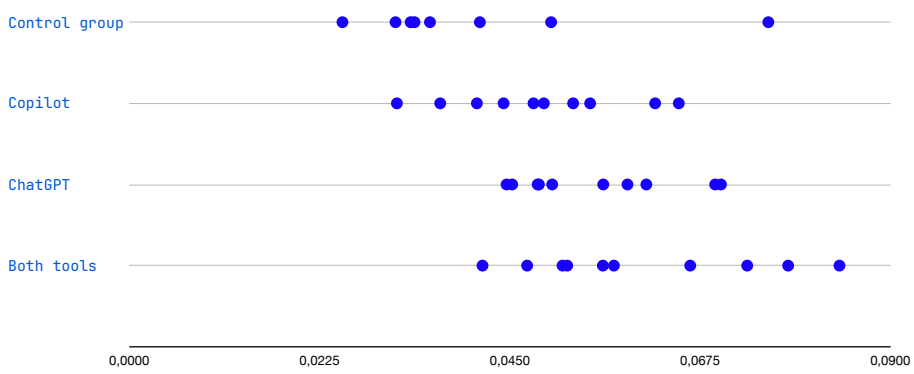

Fig. 1. Strip plot of taggings per word in journals of different groups

which we added 6053 tags as described above. Each journal received 33–446 taggings (mean 159, median 129), with a density around 5 taggings per hundred words — see Figure 1 for a strip plot of all values. Table 2 contains a summary on the number of taggings per tag per group, and Table 3 has the full data set.

4.2 Qualitative Findings

Architectural Design vs Coding Tasks. There seems to be a sweet spot for the use of GAI in programming a large project, where it could serve as a powerful assistant rather than a replacement for a critically thinking software engineer. One of the most common observations was that GAI performs poorly at high-level design and architecture — students often found that when they allowed GAI to dictate structure, the results were rigid, inefficient, or simply misaligned with their intent. As a result, many study participants settled for a hybrid approach: they took charge of the overall design process themselves but used GAI for code generation and implementation details, which often led to

Tag	Control	Copilot	ChatGPT	Both
joy	6	73	61	55
pain	67	89	128	123
boost	0	159	128	112
waste	9	122	182	133
manual	16	77	119	93
prompt	2	45	293	219
could have	0	27	24	59
would have	200	51	2	2
ai misuse	6	22	41	28
class diagram	14	19	24	15
sequence diagram	7	7	13	14
design pattern	12	20	16	19
project planning	8	14	22	7
refactoring	19	36	51	48
debugging	90	84	112	110
testing	88	164	255	136
reasoning	31	44	78	64
documentation	86	114	106	90
game rules	25	41	77	57
protocol	29	61	97	49
concurrency	32	77	72	68
network	49	65	37	56
user interaction	23	27	36	41
error handling	13	40	30	31
JML	14	12	65	15
TDD	1	1	1	1

Table 2. Total number of taggings per tag (cf. section 3) per group throughout all journals. Each of the 38 journals was written by one student participant and contains notes on the activities they were performing during the project, as well as a critical reflection on the entire process in the end.

more effective outcomes. This was to be expected for Copilot which is its original intended use, but was also quite prominent for ChatGPT.

Interestingly, some students deliberately avoided GAI for simple tasks, stating that it would likely "mess up their code" — a clear case of *code alienation*, a phenomenon well known in collaborating human developers: if some developer have strong feelings of code ownership, excessive updates to that code done by other developers or by program transformation tools (e.g., in the context of software migration and renovation), can destroy that ownership feeling. In this case, GAI-generated content apparently felt foreign or unpredictable to the developer. This was much more visible in ChatGPT-using participants, perhaps the very nature of Copilot being integrated into the IDE alleviated some of the code alienation feeling. Among participants using both tools, some reported significant success and satisfaction from synergetic use, when the design was discussed with and/or suggested by ChatGPT, after which the implementation

Tag	Control group										Copilot										ChatGPT										Both tools									
joy	2			2				2	10	5	27		12	1	11			7	2	3	21	12	5	9	2	2	3	2	4		7	3	11	9	4	16	1	
pain	13	12	11	11	4	7	4	5	1	2	20	1	22	5	6		8	24	6	10	30	38	5	18	6	8	3	4	7	2	7	2	17	25	25	23	6	9
boost									32	13	29	17	22	8	20		5	13	16	16	19	11	15	27	7	5	2	10	9	5	7	2	15	14	12	31	7	10
waste	1	4	1	1			2		13	15	20	9	10	13	15		5	22	11	7	53	27	14	18	14	13	13	12	12	8	13	3	18	12	18	32	8	9
manual	6	1	3		1	1	2	2	2	11	10	7	11	11	8	1	6	10	14	7	12	23	13	23	9	12	2	4	5	5	15		13	11	10	28	4	2
prompt	2									2	3	6	3	12	11			8	30	15	52	48	33	70	12	16	4	13	31	14	15	12	16	26	25	56	10	14
could have										4	1	3	4	3	7		1	4	1	3	1	2	4	10		3			3	4	8	1	1	1	6	29	2	4
would have	28	27	30	9	50	17	16	23	5	3	16	4	10	6	4		1	2								1		1	1					1					
ai misuse		1	2				1	2	3		3	1	5		1		1	8	9	3	2	4	4	8	9		1	1	3		1	2	9		3	5	1	4
class diagram	4	1	5		2	1	1		1	2	2		7	2	3			2	1	1	1		10	8		3			2	1			4	5	2		1
sequence diagram	2	1	2				1	1	1	2	1		2			1							12			1			1	1			7	4			1
design pattern	1	6					1	4	4		5	1	5			2	2	1	2	4	1	4		2		1		2				1	1	1	6	10		
project planning	1		1	3	1			2	1					4	5	1	1	2	1	3	4	1	1	2	6	1	2	1		1			1			2	2	1
refactoring	2	1	1	2	5	2	2	4	2	1	1	1	8	1	6	1	1	14	1		9	14	7	12	7	1			5	7	4		4	10	1	13	2	2
debugging	27	3	8	7	7	14	9	15	4	9	10	2	11	1	16	1	5	25	2	13	37	20	4	10	24	2			14	9	5	2	15	8	16	29		12
testing	1	18	10		22	10	7	20	8	9	20	10	30	14	24	24	11	14	12	4	63	35	59	39	33	7	1	2	16	7	4		15	20	19	38	7	10
reasoning	1	2	1	1	17	6		3		6	3	2	15	2	7			9	10	5	27	13	7	11	4		1		9	3	4	2	5	5	11	18	2	5
documentation	6	10	24		26		11	9	7	19	6	7	15	7	27	1	7	18	4	4	11	10	15	33	13	14		2	10	2	7	1	13	10	15	23	3	6
game rules	7	3	1	2		4	3	5	4	11	2	1	2	2	12	3	2	2	2	11	25	14	6	3	6	6	2	2	8	7	9	2	1	9	4	11	4	2
protocol	1	5	4	2	2	4	3	8	4	6	8	11	1	3	6	15	4	3	8	11	11	26	2	7	21	8	2	1	7	2	6		3	6	8	10	3	4
concurrency	1	4	3	1	8		7	8	8		6	3	10		6	21	2	21	15	1	1	7	10	9	8	15		6	6	2		4	11	4	34	5	2
network	3	7	4	15	9	5	3	3	8	4	16	12	5		3	6	5	6	4		2	15	3	7	1	3	1	1	2	2	1		1	3	13	21	8	5
user interaction	3	3	2	5	1	1	3	5	5	2	8	6			1	4		1	3	2	6	13	3	2	2	5			4		10	1	1	9	1	12	3	
error handling			2		2		2	7	2	1	10	3		2	8	6	3	5	4		10	8	5	1				2	4		3		3	1	5	6	4	5
JML		5		1			6	2					5		3			4		6	10	5	14	18	11			1	7	1	3		3			1		
TDD		1								1											1												1					

Table 3. All tagging statistics of this pilot study.

was done by Copilot with some minimal interference and/or mediation by the human.

Another emerging challenge was the gap between having a mental idea about software design and implementing it — and apparently translating it into code directly was perceived to be easier than translating it into an effective GAI prompt. This, however, might be a trainable skill which our students simply were unprepared for. Ultimately, while GAI accelerates the transition from idea to code, its effectiveness depends on the user's ability to define structure, provide clear guidance, and critically evaluate GAI-generated output. This reinforces the idea that GAI is most useful when paired with strong foundational knowledge and intentional design choices.

Supporting quotes:

- ◇ *This proves that the most effective way to use Chat GPT is to ask him to complete some missing parts/methods of the already existing code that you have created yourself.*
- ◇ *I notice that a bulk of my time is being spent on establishing architecture and design choices instead of actually programming.*
- ◇ *I did not find Chat GPT useful in general project designing, or class writing.*
- ◇ *ChatGPT does well at understanding the logic of the methods.*
- ◇ *Did not use chatGPT this hour because it was a very simple task and chatGPT may destroy what I have so far.*
- ◇ *I did not have to ask help from AI to write out the code, as I did not have any written description but had a mental idea of the structure.*
- ◇ *It did however incorrectly implement a try catch statement, in a way that Intellij wouldn't see as an error — so some knowledge on exception handling was necessary.*
- ◇ *if I hadnt known the basics of servers, I wouldn't have been able to get that.*
- ◇ *GPT only gave me a skeleton of a project that was not future proof.*

Acceleration vs Correctness. One of the most striking observations was how GAI transforms the coding workflow: study participants could simply feed requirements to a GAI tool and instantly receive functional code. In some cases, the output was immediately usable, while in many others, it provided a good foundation to refine. A similar way of working involved giving GAI an assessment rubric to generate work that would score well, which occasionally succeeded but is ultimately unrealistic for real-world software development, where predefined grading criteria do not exist, requirements tend to creep and mature solutions to rot.

The common theme among participants that this has led to, was the lack of cognitive effort required. Some students found it liberating, even enjoyable, to not have to think and still produce working code. However, this obviously comes at a price. While GAI accelerated productivity by enabling developers to generate more code faster, it also introduced uncertainty about correctness. In turn, that was leading to a growing awareness that GAI-produced solutions often require significant debugging and validation, sometimes overshadowing time won by quickly producing working code. On several occasions participants reported that after many prompts, the overall code quality declined, technical

debt accumulated rapidly, and managing GAI-induced complexity became an issue. Some found themselves completely unable to debug GAI-produced code, as it was too complex for them to fully grasp. Sometimes GAI's rapid output ended up slowing development down — one would generate large portions of code in minutes, only to spend subsequent hours debugging unexpected issues, in worst cases ultimately abandoning GAI solutions altogether and solving the problem from scratch manually. These experiences highlight an ongoing tension: while GAI reduces the barrier to writing code, it does not guarantee that the resulting code is maintainable, understandable, or even correct.

Supporting quotes:

- ◇ *First hour in, and I have just created the entire hex game.*
- ◇ [ChatGPT] *made the abstract class and an easy computer at first and let me decide how to implement it, copilot then implemented it for me.*
- ◇ *I think ChatGPT accelerates the implementation of basic methods; it is even useful for creating game logic. Correctness is the main problem.*
- ◇ *The logic from the suggestion was somewhat correct.*
- ◇ *The quality of that I didn't check.*
- ◇ *Copilot is deceptive, it looks like it is helping, but at some point or when a random bug starts popping up, you realize that the code was pretty bogus.*
- ◇ *I overspend time on trying to figure out the problem using ChatGPT.*
- ◇ *The tools help, but you have to be really careful in what you accept or don't accept.*
- ◇ *I did not have to think a second about this problem. ChatGPT solved it for me.*
- ◇ *I would have never created such a function since it is not within my knowledge.*
- ◇ *I feel like I would be spending more time debugging the code from AI than what I'm now spending on writing it on my own.*
- ◇ *When I said fix that please it gave me some optimizations which broke it completely.*
- ◇ *I have to look into the HexGame class and I do not know the code well since most of it is pasted from chatgpt.*

The Conversational Aspect. Some students claimed that their way of working would have been the same whether they did or did not have access to GAI. At least on some occasions we had to agree, up to the point where some reflective journals looked like URL lists with links to StackOverflow, Baeldung, YouTube and other sources of information, and the entire workflow was organised around seeking information or even working pieces of code to use in constructing their own system. In other cases, participants worked more in the IDE than in the browser, and only occasionally resorted to searching the web.

Yet this was always combined with anthropomorphising of the GAI: especially ChatGPT was repeatedly referred to with male pronouns (he/him), and the choice of verbs — e.g., "he forgot", "it struggles a lot", "he knows", "it decided/refused to implement" or "he makes no sense" — indicated perception of these prompt and response exchanges as true conversations. GAI became more than just a tool and turned into a discussion partner.

This ability to use GAI to not just generate code from natural language prompts, but also to discuss strong or weak points of envisioned design, allowed to overcome the challenges we named in the first theme of this section. Having a

conversation partner with deep knowledge of concrete details of using pretty much any library and API in existence and being capable to maintain a conversation about any software design topic from algorithm design to code quality, also led to contradicting the second theme (acceleration and correction). Some participants expressed strong beliefs in correctness of GAI-produced code beyond the faith they had in their own.

One of the three most common regrets found in the **would have** category (the other two will be discussed in the next paragraph), is the lack of option to request an explanation. At least some of the representatives of the control group as well as the Copilot users followed the same process of "prompt-based programming" as one would do with GAI, just by using web search, reading posts on StackOverflow and guides on Baeldung, calling up friends, chatting them on Discord, etc, and regretted that getting to know some API or a language feature requires them to watch a long tutorial video instead ChatGPT just spoonfeeding that information directly.

Supporting quotes:

- ◇ *Normally I would ask ChatGPT how to generate this, but since I could not, I resorted to watching a YouTube video.*
- ◇ *No doubt stackoverflow had something similar.*
- ◇ *I'm basically just copy pasting solutions from the internet.*
- ◇ *I am not sure if AI could help me with this, but of course using AI will prevent bugs from being created in the first place.*
- ◇ *when I ask to compare different ways to implement a functionality, it provides a pros and cons list on how it would affect the product.*
- ◇ *I deleted my program code and Copilot was able to just fill in the code as it always wanted to.*
- ◇ *It was not of much help, because I didn't trust it to give me the correct answer since its answers seemed vague.*
- ◇ *Right now, I understand the code perfectly.*

Where GAI Helps the Most. There are three reasonably well-defined domains where GAI was perceived as being the most helpful both by participants who did use it, as well as by those who were not given that option. The first two strongly correspond to the remaining two categories of popular complaints around the **would have** tag: testing and documentation.

The perception of GAI's usefulness in testing appears to be largely illusory rather than substantiated by concrete benefits. The ultimate purpose of software testing is not to simply have a test suite with some high enough coverage of existing code, but to use it to find bugs or to prevent those bugs from entering the code in the future. While many participants — both those who used GAI tools and those who did not — eagerly recognised the potential of GAI-produced test cases, the data we have seen suggests that this perception is not grounded in actual problem-solving instances. Nowhere in the reflective journals nor in focus group discussions did students report a case where a GAI-produced test case identified or prevented a bug that would not have been caught otherwise. Instead, the tests produced by GAI tend to appear comprehensive and sophisticated,

inducing a sense of false security, and often incorporating a variety of edge cases and structured assertions. However, in practice, these test cases are the antitheses of TDD: they are green on arrival, since they pass successfully immediately upon generation, and they stay green throughout their lives, rarely contributing to meaningful debugging. This suggests that while GAI can generate plausible-looking tests, it does not inherently improve the iterative debugging process, and students may overestimate their effectiveness simply because they resemble well-structured test suites. In scenarios where GAI was used to take one or two manually written test cases to generalise and produce similar test cases covering slightly different but related methods, it could be argued that the GAI application was useful and justified.

Generating documentation, be it in the form of comments in the code, user manuals or structured documentation formats such as Javadoc or JML, appears to be a more promising application of GAI. Unlike GAI-produced test cases, which often serve more as a decorative layer than a functional tool for debugging, documentation produced by ChatGPT and GitHub Copilot exhibits a greater potential for practical utility. Participants frequently reported that GAI-assisted documentation helped them articulate design decisions better than they would have done themselves, and generate boilerplate descriptions that they could then refine further (or, in some cases, especially for Copilot, to go the other way and generate code from an extensive Javadoc).

One possible explanation for this difference lies in the nature of GAI's training data. GAI models struggle to generate truly novel test cases because it is a substantial reasoning challenge to find corner cases that truly need testing. Yet, they have been trained on large corpora which included lots of well-written technical documentation, including open-source repositories and software manuals, conforming to industrial best practices. As a result, ChatGPT in particular often produces coherent, contextually appropriate explanations, sometimes even suggesting JML constraints that align with best practices in formal verification. However, this remains an area that requires further research. Initial impressions do suggest that GAI-produced documentation is more than just surface-level filler, but we still need to investigate whether students actually understand and internalise what is being generated, or if they are merely accepting GAI outputs at face value. It is a challenge moving forward to establish whether GAI-produced documentation enhances learning and software quality, or whether it simply creates a false sense of completeness, much like GAI-produced test suites.

Interestingly, some participants of our study believed that the biggest gains from GAI they had or could have had, came from implementing the game rules themselves. The reasons might be at least partly algorithmic, since the game we have used in this study, used a hexagonal board, and it takes quite some design and thinking ahead to figure out what it means for places on such a grid to be adjacent to one another or how to construct a path from one place to another. The other explanation could be similar to a writer's block: even though some implementation parts are relatively easy and straightforward, it can be considerably easier to start with some basic setup already coded, and fill

in the blanks incrementally, rather than to start from scratch by creating first classes one by one. It requires much more data points that we could obtain, to be conclusive here (and to account for confounding factors like the choice of a programming language — since it is also possible that starting a new project in a scripting multiparadigm language like Python would be less of a mental burden).

Supporting quotes:

- ◇ *Chat GPT is a very useful tool for writing texts, providing information to a concern in a matter of seconds and debugging code.*
- ◇ *I have no idea how that JML part works and I have no idea what it has written down in the JavaDoc.*
- ◇ *I do believe that without the AI, the hardest part of the project would have been the gamelogic.*
- ◇ *On the report side of things, I found GPT to be a lifesaver. I practically made GPT generate all of my report.*
- ◇ *for the report side of things, it would be like getting homework from your friend. All you have to do is change it a little bit and the work is completed.*
- ◇ *Copilot does not help me think, it helps get rid of brainless typing.*
- ◇ *This took like 40 minutes to complete 4 diagrams. I am shocked because I remember this taking me a very long time 2 years ago.*
- ◇ *I solved the problems above myself because I did thought chatgpt would be more time consuming.*

Hallucinations are situations when GAI makes assumptions about the context which were neither provided directly by the user, nor can be derived from the prompts provided by the user. Initially observed in the computer vision domain where they were seen as a positive phenomenon, allowing creative movie-like zooming of low quality images by guessing and filling in content that was inherently missing [3], nowadays hallucinations are synonyms for confabulations, fabrications, delusions and other forms of falsification. We list four important subcategories here, referring to the work of Maleki et al for a recent literature review on the matter [29].

Confabulations are GAI's responses which are plausible but not factually correct. For code generation, a confabulation would be producing method calls to methods that do not exist even though it would make sense to have methods with those names. **Fabrications** are GAI's responses where inaccurate information is added to the context without underlying reasoning and presented as factual. For code generation, a fabrication could lead to a GAI tool suddenly switching to a different programming language in the middle of the session simply because the training data indicates it more probable to have this algorithm implemented in a language other than what was requested. **Delusions** are responses which are blatantly wrong, which happens often in the context of debugging: in order to advise the user on how to debug a certain problem, the GAI tool needs to come up with a hypothesis about the possible cause of the observed faulty behaviour, and since that can be far off (in addition to not being communicated properly), it is known to send users on a wild goose chase. Finally, there is **parroting** which stems from repeating patterns commonly observed in the training data

— in particular dangerous in the context of programming education due to its tendency to not only easily solve all introductory programming exercises, but also to just as easily inject common mistakes into such solutions, combining them with very believable justifications.

Supporting quotes:

- ◇ (C) *And then I spotted another assumption that was made by the AI and it was wrong.*
- ◇ (F) *It seemed like a good solution, but when I asked it to implement it for me, it was implemented in Python.*
- ◇ (F) *apparently Copilot thought I was making Checkers.*
- ◇ (C/D) *Much other stuff is not exactly right.*
- ◇ (D) *I started to believe that the more questions I ask, the dumber it gets.*
- ◇ (D) *Also, some tests didn't really test what they were supposed to test.*
- ◇ (P) *I do notice that my code does look very bad now I use chatGPT.*

Learning experience By now the paradigm shift in how students approach programming tasks within a project when using GAI tools, should be obvious. The key question still remains, and it is not whether students will use these tools in the future — because they inevitably will — but rather whether they truly learn anything from them or while using them. Do students gain a deeper understanding of programming concepts, or does GAI obscure the cognitive processes necessary for effective learning? Do we, as educators, need to redefine what we teach, or simply adjust how we teach it?

Traditionally, project-based learning [15] has been rooted in active engagement: students learn by struggling through problems by themselves, making mistakes, debugging, iterating on their solutions, and getting timely actionable feedback to let them to keep going. The introduction of GAI shifts the balance: students are now less engaged in the *doing* and more engaged in the guiding of a GAI tool to do it for them. Such students are focused on the ultimate goal — getting the project to work, submitting it on time and getting a good grade — even though for teachers this has never been the real goal, and within the learning-by-doing paradigm the project deliverable is always secondary to the process of doing it.

We did have participants that reported that they did not feel they missed much from the learning experience, strongly suggesting that GAI merely accelerated parts of the project rather than replacing genuine problem-solving. Others, however, noted that they had a *false* sense of efficiency, since they assumed AI would make things easier, only to realise later that they ended up with code they do not understand and hence have no true control over. The most recurring theme was one of over-reliance: students who leaned too heavily on GAI tools, often found themselves unable to reason about their own code whenever problems arose. Ironically, proceeding that way makes them go the full circle, since spending too much time on understanding or debugging GAI code the first time, makes them less eager to rely on it for anything in the next steps in fear that their productivity will take another hit.

In this sense, GAI introduces a new failure mode in programming education: instead of struggling with syntax and logic, students struggle with interpreting

GAI-produced solutions. This is a new kind of problem, one that traditional programming education does not prepare students for. However, this is far from a universal experience. Some students did learn from GAI, especially when they approached it with a genuine desire to understand. Those who treated GAI as a tutor rather than an oracle — questioning its responses, verifying its correctness, iterating on its suggestions, persisting in demanding justifications and detailed adjustments — reported a meaningful learning experience.

Supporting quotes:

- ◇ *I don't think I missed too much in terms of the learning experience.*
- ◇ *Very easy to trust ChatGPT too much, which I have done again.*
- ◇ *I realized that working with ChatGPT is actually quite difficult.*
- ◇ *I simply have no idea how to even ask chatGPT for advice, as that would take explaining my entire codebase.*
- ◇ *I also have no idea how the function works so I will have to check that to fix it. Maybe I should ask chatGPT again.*
- ◇ *This is probably because I thought it would be quick an easy using AI, without it I would have first assessed what was actually necessary.*
- ◇ *Maybe I have become over reliant on the tool, because not using it probably made this go way faster.*
- ◇ *It's much faster to do this without AI and I get better results.*
- ◇ *I didn't try asking ChatGPT to do that, as I considered it would have taken me more time to explain exactly what I wanted, and even then it would have probably required a lot of changes from my side.*
- ◇ *I would thus think that Copilot did not prevent me from learning about coding it just took over the easy duties while leaving the more complicated work for me to figure out.*
- ◇ *I mostly did not have to think at all which did have some drawbacks.*
- ◇ *Coding wise I did not learn much, I do not know a lot about my own project.*
- ◇ *You can learn from chatgpt if you genuinely want to.*
- ◇ *This is definitely not good practice, but let's hope that this works.*

5 Implications

5.1 GAI as a Tutor

One of the most immediate and apparent uses of GAI in education is as an on-demand tutor, capable of providing explanations, clarifications, technical details at any time of day. Unlike human instructors, GAI does not have office hours, does not get tired, impatient, always delivers responses within seconds or minutes, and never complains. This convenience is undeniably valuable, and students struggling with a difficult concept or an unfamiliar API can quickly obtain an explanation without waiting for a scheduled class or response from a peer. The ability to receive immediate feedback allows for just-in-time learning, potentially making students more independent and resourceful.

However, students have traditionally relied on teachers and textbooks as authoritative sources, and they still expect correctness from those they learn from. GAI models do not always distinguish between correct, misleading, and outright

false information, and in sections above we provided much evidence of all these expectations being violated. A student who assumes that GAI-produced responses are always accurate, may unknowingly build a foundation of misconceptions, particularly when GAI confidently hallucinates plausible-sounding but incorrect explanations. If we assume that human tutors know their students and match their level of understanding as well as increasingly probe them for misconceptions, then GAI in its current forms fails quite decisively, lacking the ability to take enough context into account, be it about a particular student's needs or about "remembering" previous conversations and consistently building up from those in an informed and pedagogically sound way.

Beyond accuracy concerns, the ease of access to information may fundamentally alter how students engage with learning. Just as widespread calculator use has made people less proficient at mental arithmetic, and reliance on digital devices has diminished handwriting skills, continuous GAI-assisted explanations could lead to a decline in deep knowledge acquisition. Students may grow accustomed to looking up facts and reasoning rather than internalising them, reducing their ability to recall and apply concepts without GAI assistance. This could lead to a generation of programmers who can efficiently retrieve information but struggle to reason through problems independently. Some claim that this milestone has already been reached with developers who spend more working hours on StackOverflow and YouTube than in the actual editor, but extensive GAI tutoring institutionalises this.

Despite these concerns, GAI as a tutor has clear potential to enhance learning when used effectively. A student who actively engages with GAI explanations, cross-references multiple sources, and critically evaluates responses, may actually develop a stronger conceptual foundation than one relying solely on lectures and textbooks. The key difference will likely be in how GAI is used rather than whether it is available, and further research will be needed to determine whether it ultimately enhances or erodes understanding in the long run.

5.2 GAI as a Tool

In the ethics research community, the discussion whether GAI is "just a tool" or rather "more than a tool", is still ongoing [2]. However, if for the purpose of this section we assume that it is just a powerful tool, and we keep the educational model which relies on a guidance of a human instructor, then GAI would not replace the teacher but instead augments the learning process, much like an IDE that speeds up routine tasks like performing simple refactorings and provides immediate feedback in contexts like regression testing. This shifts the nature of programming education: not by making traditional coding obsolete, but by requiring a new set of complementary skills. Just as using a calculator does not eliminate the need for mathematical reasoning, leveraging GAI tools does not mean students can ignore fundamental programming concepts. However, the way they engage with those concepts, may change significantly.

GAI tools, just like powerful IDEs, are tools not just for making code run, like compilers are, or tools for keeping track of the code, like version control

systems. In addition to all that, they are decision-making aids, influencing how developers (and students as future developers) approach problems, how they structure their code, and even how they understand the boundaries of what is possible altogether. Because of this, GAI literacy becomes an essential skill. Students must learn to critically assess GAI-produced suggestions, recognise when they are incorrect or inefficient, refine their prompts to get meaningful results. Prompt engineering as the ability to formulate effective queries that yield useful GAI responses, is already becoming a crucial skill in the industry, and education must adapt accordingly. Ignoring this reality would be like insisting that students write assembly code or use punch cards when the world has long since moved to high-level languages and colour displays.

Modern software developers do not need to know assembly to be competent programmers, and many never write raw SQL because ORMs abstract away the details. If GAI-assisted coding becomes the norm, should we really resist it, or should we accept that programming is evolving? There is an argument to be made that, as long as students develop an understanding of problem-solving, software architecture, and debugging, the method of implementation might be secondary after all. There is still plenty to teach beyond writing code: understanding what to build, why certain approaches work, how to make existing systems better, etc. Even responsible GAI usage needs to be explicitly taught. Without structured guidance, students will inevitably use GAI anyway, but without learning where to trust it, how to verify it, and when to rely on their own skills instead. If we fail to integrate GAI into education, we do not prevent students from using it, we only ensure that they use it uncontrollably.

5.3 Skills GAI Cannot Replace

If GAI-assisted programming is here to stay, then perhaps education should specifically focus on what GAI cannot do. While GAI tools can certainly generate code, explanations, and even documentation, it does not reason in a critical way, verify correctness, or make decisions based on deeper understanding. If students are to become effective programmers, they must develop these skills themselves, not by passively consuming AI outputs, but by actively questioning, refining, and integrating them into a broader problem-solving process.

One key area where we found this to be evident, is documentation. Many students in our study postponed writing any kind of documentation (not just the report, but also code comments and Javadoc specifications) until the very end of the project, treating it as a separate, optional step to improve their grades, rather than an integral part of the development process. GAI makes it easy to generate documentation, but this automation hides its true purpose. Of course, documenting code is not just about explaining it to others, but mostly about clarifying ideas for oneself. When developers write documentation alongside their code, they engage in an essential cognitive process: articulating design choices, recognising inconsistencies, ensuring that future modifications remain manageable. If students only document their work at the last minute, whether manually or

via GAI, they miss the opportunity to use documentation as a real-time thinking aid rather than a retroactive justification.

The same applies to testing, which very much fell under the same bus. When done at the end of a project, testing serves only as an *a posteriori* bug-finding mechanism, revealing last-minute issues that must be patched before deployment. However, when testing is integrated throughout development, it becomes a tool for confidence and flexibility. Writing tests early allows programmers to refactor their code more freely, knowing that mistakes will be caught before they become deeply embedded. GAI can certainly generate test cases, but if students do not internalise why testing matters and at which point in their project is it meaningful, they risk falling into the same pattern of only testing at the last minute, rather than using tests as a way to find bugs early and guide development from the start.

Perhaps the most future proof skill is debugging GAI mistakes. Traditional debugging involves tracing logical errors, identifying faulty assumptions, and methodically isolating the source of a problem [55]. GAI-assisted programming introduces an entirely new category of debugging: correcting errors in GAI code that looks correct but is subtly wrong. This is an advanced skill, requiring not just programming knowledge but an understanding of how GAI makes mistakes, how to recognise hallucinations, and how to verify GAI outputs instead of blindly trusting them. Without explicit training, students struggle to identify these issues, much like novice programmers initially struggle with debugging their own mistakes.

There are enough skills we can cover in programming education, even if all the ones where GAI has any significant success, are eliminated. Self-explanation, project and time management, structured thinking, correctness verification, assumption validation, detecting faulty reasoning, and many others. This could be seen as an opportunity to redirect our focus to produce relevant experts in the GAI-driven world.

5.4 GAI and Assessment

The first detected instances of a student producing GAI-generated code, were instinctively considered academic misconduct: you did not write this code, you do not deserve the grade, come back and redo, or else. Our study has provided a lot of evidence in support of that. For instance, we see that GAI tools do impact the grades in a significant way (with Copilot having the most impact on getting higher grades for the code, and ChatGPT having the most impact on the quality of the documentation). As another concrete example, we see that if given a chance, students generate project documentation, including drawing diagrams, instead of making it manually, and do that at the last possible moment. This supports that the assessment methods of the pre-GAI world, where students were expected to independently produce every line of code, every report, and every test case, themselves, are based on an obsolete assumption.

Declaring any use of GAI as academic misconduct is not a viable solution. Students will use GAI regardless of official policies, just as they already use

StackOverflow, online documentation, automated debugging tools, plugins. Blanket prohibitions are not only unenforceable but also misguided, since GAI is quickly becoming an integral part of professional software development, and can be considered an important career skill. Thus, the position of many teachers has gradually shifted towards a much more timid demand that one must *understand* the code they submit, or be able to *explain* how it works, or any combination of those.

If our module is called *Software Systems*, ideally we would like to test whether students produced a well designed, cleanly coded, functional solution, which they can also maintain and evolve. We would like to assign higher grades to students who excel at those criteria, not necessarily differentiating between those that used more or less tools to arrive at that skillset. This is not a new issue: in group projects, for example, freeriding [34] has always been a challenge, where weaker students can benefit from stronger teammates without contributing equally. GAI extends this problem beyond group work: now, an individual student can appear more competent than they actually are, simply by prompting GAI to do the hard work for them.

Some possible alternative assessment methods include:

- ◇ Oral exams, where students must defend their code, explaining design choices and debugging steps.
- ◇ Project exams, where students answer questions about their submitted work, ensuring they understand the details.
- ◇ In-class coding tasks, where students work under controlled conditions, demonstrating their ability to think through problems independently.
- ◇ Live demonstrations and walkthroughs, requiring students to articulate their approach and showcase their decision-making process.

These approaches shift the focus away from final deliverables and toward process, reasoning and understanding. Rather than assessing whether a project "just works", exams and interactive assessments ensure that students have actually engaged with and learned from the experience. Many institutions are experimenting now with these assessment forms, as well as with a range of university-level and programme-level policies. At least some of those are simply transparency requirements, demanding that students disclose when and how they used GAI, which is a very legalese, shortsighted and bureaucratic way, since it again takes away the focus from learning how to make a good software system to learning how to rehearse the steps of using one of the many tools in one's toolbox. Future proof rethinking of educational programming projects may focus less on writing raw code from scratch and more on interpreting, refining, debugging and integrating GAI-generated solutions. The role of the educator, then, shifts from evaluating code quality to evaluating how students engage with GAI responsibly, which we covered in the previous sections.

6 Conclusion

Our study concerned the impact of GAI tools on programming education, specifically in the context of project-based learning which implies large holistically assessed take-home assignments. Our general conclusion is that students who had access to OpenAI ChatGPT and GitHub Copilot, generally produced higher-quality code and documentation, but their learning experience was noticeably altered. While some claimed that GAI simply accelerated their work without diminishing their understanding, others found themselves struggling to engage meaningfully with the GAI-produced code, raising concerns about their learning and cognitive development.

One of the most significant takeaways is that GAI use is unavoidable. When explicitly permitted, students engaged fully and explored various ways to utilise GAI tools. When not permitted, they regretted the restriction — Copilot users tended to miss some features only provided by ChatGPT, and control group missed both; Copilot was never explicitly missed in student journals, but repeatedly reported as useful even in the presence of ChatGPT. Participants of the control group loathed their limitations and resorted to emulating the same experience with Google Search, StackOveflow and similar pre-GAI-era means. This suggests that educational institutions must adapt assessment methods, ensuring that learning is measured through understanding, reasoning, and process, rather than simply evaluating final project outcomes. Traditional grading models are increasingly ill-suited for a world where GAI can generate functionally correct solutions with minimal human intervention.

Instead of resisting GAI, programming education must shift focus toward teaching what GAI cannot do fully: critical thinking, debugging, testing, structured problem-solving, software design, reasoning and verification. We hope that the detailed breakdown of our findings in subsection 4.2 will support our colleagues and fellow educators in making informed decisions and taking sensible steps towards future proof education and assessment.

Ultimately, GAI is neither a shortcut nor a replacement for learning. It is a tool that can either enhance or erode education, depending on how it is integrated. If used thoughtfully, it has the potential to make students faster, more efficient, and better-prepared for a career in the software industry. If used recklessly, it risks producing programmers who can generate code but cannot understand, debug, or improve it. The future of programming education will depend on how well we strike this balance, ensuring that GAI serves as an amplifier of human intelligence rather than a crutch that weakens it. Much larger studies are needed as further research to refine strategies for GAI integration, assess long-term impacts on learning outcomes, and ensure that our education continues to produce competent software engineers.

References

1. Admiraal, C., van den Brink, W., Gerhold, M., Zaytsev, V., Zubcu, C.: Deriving Modernity Signatures of Codebases with Static Analysis. Special Issue in the Journal

of Systems and Software: Open Science in Software Engineering Research (JSS) **211** (May 2024). https://doi.org/10.1016/j.jss.2024.111973
2. Babushkina, D.: Are We Justified Attributing a Mistake in Diagnosis to an AI Diagnostic System? AI Ethics **3**(2), 567–584 (2023). https://doi.org/10.1007/S43681-022-00189-X
3. Baker, S., Kanade, T.: Hallucinating Faces. In: Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition. pp. 83–88 (2000). https://doi.org/10.1109/AFGR.2000.840616
4. Bender, E.M., Gebru, T., McMillan-Major, A., Shmitchell, S.: On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? In: Proceedings of the Conference on Fairness, Accountability, and Transparency. pp. 610–623. FAccT, ACM (2021). https://doi.org/10.1145/3442188.3445922
5. Bommasani, R., Hudson, D.A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M.S., Bohg, J., Bosselut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N., Chen, A., Creel, K., Davis, J.Q., Demszky, D., Donahue, C., Doumbouya, M., Durmus, E., Ermon, S., Etchemendy, J., Ethayarajh, K., Fei-Fei, L., Finn, C., Gale, T., Gillespie, L., Goel, K., Goodman, N., Grossman, S., Guha, N., Hashimoto, T., Henderson, P., Hewitt, J., Ho, D.E., Hong, J., Hsu, K., Huang, J., Icard, T., Jain, S., Jurafsky, D., Kalluri, P., Karamcheti, S., Keeling, G., Khani, F., Khattab, O., Koh, P.W., Krass, M., Krishna, R., Kuditipudi, R., Kumar, A., Ladhak, F., Lee, M., Lee, T., Leskovec, J., Levent, I., Li, X.L., Li, X., Ma, T., Malik, A., Manning, C.D., Mirchandani, S., Mitchell, E., Munyikwa, Z., Nair, S., Narayan, A., Narayanan, D., Newman, B., Nie, A., Niebles, J.C., Nilforoshan, H., Nyarko, J., Ogut, G., Orr, L., Papadimitriou, I., Park, J.S., Piech, C., Portelance, E., Potts, C., Raghunathan, A., Reich, R., Ren, H., Rong, F., Roohani, Y., Ruiz, C., Ryan, J., Ré, C., Sadigh, D., Sagawa, S., Santhanam, K., Shih, A., Srinivasan, K., Tamkin, A., Taori, R., Thomas, A.W., Tramèr, F., Wang, R.E., Wang, W., Wu, B., Wu, J., Wu, Y., Xie, S.M., Yasunaga, M., You, J., Zaharia, M., Zhang, M., Zhang, T., Zhang, X., Zhang, Y., Zheng, L., Zhou, K., Liang, P.: On the Opportunities and Risks of Foundation Models (2022), https://arxiv.org/abs/2108.07258
6. Boughattas, N., Neji, W., Ziadi, F.: Project Based Assessment in the Era of Generative AI-Challenges and Opportunities. In: Proceedings of the 20th International CDIO Conference. pp. 347–356 (2024), https://www.cdio.org/sites/default/files/documents/360_CDIO%202024%20Proceedings.pdf
7. Brockman, G., Lightcap, B., Murati, M., Clark, C.: OpenAI ChatGPT. https://chat.openai.com/chat/ (2022)
8. Burdy, L., Cheon, Y., Cok, D.R., Ernst, M.D., Kiniry, J.R., Leavens, G.T., Leino, K.R.M., Poll, E.: An Overview of JML Tools and Applications. International Journal on Software Tools for Technology Transfer **7**(3), 212–232 (Jun 2005). https://doi.org/10.1007/s10009-004-0167-4
9. Cao, H., Tan, C., Gao, Z., Xu, Y., Chen, G., Heng, P.A., Li, S.Z.: A Survey on Generative Diffusion Models. IEEE Transactions on Knowledge and Data Engineering **36**(7), 2814–2830 (2024). https://doi.org/10.1109/TKDE.2024.3361474
10. Ciniselli, M., Pascarella, L., Bavota, G.: To What Extent do Deep Learning-based Code Recommenders Generate Predictions by Cloning Code from the Training Set? (2022), https://arxiv.org/abs/2204.06894
11. Clandinin, D.J., Caine, V.: Narrative Inquiry. In: Reviewing Qualitative Research in the Social Sciences, pp. 178–191. Routledge (2013). https://doi.org/10.4135/9781412963909.n275

12. Denny, P., Luxton-Reilly, A., Tempero, E., Hendrickx, J.: CodeWrite: Supporting Student-Driven Practice of Java. In: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education. p. 471–476. SIGCSE, ACM (2011). https://doi.org/10.1145/1953163.1953299
13. Farooq, A., Zaytsev, V.: There Is More Than One Way to Zen Your Python. In: Visser, E., Kolovos, D., Söderberg, E. (eds.) Proceedings of the 14th International Conference on Software Language Engineering (SLE). pp. 68–82. ACM (2021). https://doi.org/10.1145/3486608.3486909
14. Freise, L.R., Bruhin, O., Ritz, E., Li, M.M., Leimeister, J.M.: Code and Craft: How Generative AI Tools Facilitate Job Crafting in Software Development. Available at SSRN 4974037 (2024), https://ssrn.com/abstract=4974037
15. Gary, K.: Project-Based Learning. Computer **48**(9), 98–100 (2015). https://doi.org/10.1109/MC.2015.268
16. GitHub: GitHub Copilot — Your AI Pair Programmer. https://github.com/features/copilot (2021)
17. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative Adversarial Networks. Communications of the ACM **63**(11), 139–144 (Oct 2020). https://doi.org/10.1145/3422622
18. Höög, O., Ljungqvist, P.: Competence Dynamics in the Age of AI: A Qualitative Study Examining How Generative AI Alters the Relevance and Need of Competences among High-Skilled Workers. Master's thesis, Graduate School, School of Business, Economics and Law, University of Gothenburg (2024), https://hdl.handle.net/2077/82431
19. Hough, L.: Truce Be Told. Harward Ed. https://www.gse.harvard.edu/ideas/ed-magazine/11/09/truce-be-told (Sep 2011)
20. Hytti, H., Takalo, R., Ihalainen, H.: Tutorial on Multivariate Autoregressive Modelling. Journal of Clinical Monitoring and Computing **20**(2), 101–108 (Apr 2006). https://doi.org/10.1007/s10877-006-9013-4
21. Kallick, B., Zmuda, A.: Orchestrating the Move to Student-Driven Learning. Educational Leadership **74**(6), 53–57 (2017), https://www.learningpersonalized.com/wp-content/uploads/2017/07/Article-Orchestrating-the-Move_KallickZmuda.pdf
22. Kingma, D.P., Welling, M.: Auto-Encoding Variational Bayes (2022), https://arxiv.org/abs/1312.6114
23. Kokotsaki, D., Menzies, V., Wiggins, A.: Project-Based Learning: A Review of the Literature. Improving Schools **19**(3), 267–277 (2016). https://doi.org/10.1177/1365480216659733
24. Kramer, D.: API Documentation from Source Code Comments: A Case Study of Javadoc. In: Proceedings of the 17th Annual International Conference on Computer Documentation. p. 147–153. SIGDOC, ACM (1999). https://doi.org/10.1145/318372.318577
25. Kusam, V.A.: Generative-AI Assisted Feedback Provisioning for Project-based Learning in CS Education. Master's thesis, University of Michigan–Dearborn, Dearborn, USA (2024). https://doi.org/10.7302/22651
26. Kusuma, J.S., Halim, K., Pranoto, E.J.P., Kanigoro, B., Irwansyah, E.: Automated Essay Scoring Using Machine Learning. In: Proceedings of the Fouth International Conference on Cybernetics and Intelligent System. pp. 1–5. ICORIS (2022). https://doi.org/10.1109/ICORIS56080.2022.10031338
27. Lau, S., Guo, P.: From "Ban It Till We Understand It" to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot.

In: Proceedings of the ACM Conference on International Computing Education Research — Volume 1. p. 106–121. ICER, ACM (2023). https://doi.org/10.1145/3568813.3600138
28. Leung, J.K.L.: Applied Generative AI for Interdisciplinary Projects. Effective Practices in AI Literacy Education: Case Studies and Reflections pp. 179–188 (2024). https://doi.org/10.1108/978-1-83608-852-320241019
29. Maleki, N., Padmanabhan, B., Dutta, K.: AI Hallucinations: A Misnomer Worth Clarifying. In: Proceedings of the IEEE Conference on Artificial Intelligence (CAI). pp. 133–138 (2024). https://doi.org/10.1109/CAI59869.2024.00033
30. Mastropaolo, A., Pascarella, L., Guglielmi, E., Ciniselli, M., Scalabrino, S., Oliveto, R., Bavota, G.: On the Robustness of Code Generation Techniques: An Empirical Study on GitHub Copilot . In: Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE). pp. 2149–2160. IEEE Computer Society (May 2023). https://doi.org/10.1109/ICSE48619.2023.00181
31. Newkirk, J., Vorontsov, A.A.: Test-Driven Development in Microsoft .NET. Microsoft Press, Redmond, WA (2004)
32. Niszczota, P., Conway, P.: Judgments of Research Co-created by Generative AI: Experimental Evidence (2023), https://arxiv.org/abs/2305.11873
33. O'Grady, M.J.: Practical Problem-Based Learning in Computing Education. ACM Transactions on Computing Education (TOCE) **12**(3) (Jul 2012). https://doi.org/10.1145/2275597.2275599
34. Palfrey, T.R., Rosenthal, H.: Testing Game-Theoretic Models of Free Riding: New Evidence on Probability Bias and Learning. Cambridge, Mass.: Dept. of Economics, Massachusetts Institute of Technology (1990), https://dspace.mit.edu/bitstream/handle/1721.1/64219/testinggametheor00palf.pdf
35. Papamakarios, G., Pavlakou, T., Murray, I.: Masked Autoregressive Flow for Density Estimation. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017), https://proceedings.neurips.cc/paper/2017/hash/6c1da886822c67822bcf3679d04369fa-Abstract.html
36. Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., Karri, R.: Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. Communications of the ACM **68**(2), 96–105 (Jan 2025). https://doi.org/10.1145/3610721
37. Pesovski, I., Santos, R., Henriques, R., Trajkovik, V.: Generative AI for Customizable Learning Experiences. Sustainability **16**(7) (2024). https://doi.org/10.3390/su16073034
38. Pudari, R., Ernst, N.A.: From Copilot to Pilot: Towards AI Supported Software Development (2023), https://arxiv.org/abs/2303.04142
39. Pudasaini, S., Miralles-Pechuán, L., Lillis, D., Llorens Salvador, M.: Survey on AI-Generated Plagiarism Detection: The Impact of Large Language Models on Academic Integrity. Journal of Academic Ethics (Nov 2024). https://doi.org/10.1007/s10805-024-09576-x
40. Qadir, J.: Engineering Education in the Era of ChatGPT: Promise and Pitfalls of Generative AI for Education. In: 2023 IEEE Global Engineering Education Conference (EDUCON). pp. 1–9 (2023). https://doi.org/10.1109/EDUCON54358.2023.10125121
41. Ranzato, M., Boureau, Y.L., Chopra, S., LeCun, Y.: A Unified Energy-Based Framework for Unsupervised Learning. In: Meila, M., Shen, X. (eds.) Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 2, pp. 371–379. PMLR (Mar 2007), https://proceedings.mlr.press/v2/ranzato07a.html

42. Rump, A., Zaytsev, V., Mader, A.: Requirements for an Automated Assesment Tool for Learning Programming by Doing. In: Proceedings of the 18th IEEE International Conference on Software Testing, Verification and Validation (ICST) (2025)
43. Sajadi, S., Huerta, M., Ryan, O., Drinkwater, K.: Harnessing Generative AI to Enhance Feedback Quality in Peer Evaluations within Project-Based Learning Contexts. International Journal of Engineering Education (2024), https://www.ijee.ie/1atestissues/Vol40-5/02_ijee4488.pdf
44. van Santen, J.: Using LLM Chatbots to Improve the Learning Experience in Functional Programming Courses. Bachelor's thesis, Universiteit Twente, Enschede, The Netherlands (Feb 2024), http://purl.utwente.nl/essays/98155
45. Shaer, O., Cooper, A.: Integrating Generative Artificial Intelligence to a Project-Based Tangible Interaction Course. IEEE Pervasive Computing **23**(1), 63–69 (2024). https://doi.org/10.1109/MPRV.2023.3346548
46. Tamilselvi, C., Maanu P, A., Priya T, A., Kalaiyarasi, R., Nithiyasree P., Mohanaprakash T. A.: Empowering Coders: Revolutionizing Programming Education with NLP and Challenge-Based Learning. In: Proceedings of the Third International Conference on Smart Technologies and Systems for Next Generation Computing. pp. 1–6. ICSTSN (2024). https://doi.org/10.1109/ICSTSN61422.2024.10670818
47. University of Twente: *Learning-by-Interacting: The University of Twente Vision on Learning and Teaching*. https://www.utwente.nl/en/service-portal/organisation-regulations-and-codes-of-conduct/vision-on-learning-and-teaching (Apr 2023)
48. Vahid, F.: AI in CS Education: Opportunities, Challenges, and Pitfalls to Avoid. ACM Inroads **15**(3), 52–57 (Aug 2024). https://doi.org/10.1145/3679205
49. Vargas-Murillo, A.R., Pari-Bedoya, I.N.M. de la A., Guevara-Soto, F. de J.: The Ethics of AI Assisted Learning: A Systematic Literature Review on the Impacts of ChatGPT Usage in Education. In: Proceedings of the 2023 8th International Conference on Distance Education and Learning. p. 8–13. ICDEL, ACM (2023). https://doi.org/10.1145/3606094.3606101
50. Walters, W.H.: The Effectiveness of Software Designed to Detect AI-Generated Writing: A Comparison of 16 AI Text Detectors. Open Information Science **7**(1), 20220158 (2023). https://doi.org/10.1515/opis-2022-0158
51. Willis, S., Byrd, G., Johnson, B.D.: Challenge-Based Learning. Computer **50**(7), 13–16 (2017). https://doi.org/10.1109/MC.2017.216
52. Wood, D.F.: Problem Based Learning. British Medical Journal **326**(7384), 328–330 (2003). https://doi.org/10.1136/bmj.326.7384.328
53. Wu, T., Chang, M.: Application of Generative Artificial Intelligence to Assessment and Curriculum Design for Project-Based Learning. In: Proceedings of the International Conference on Engineering and Emerging Technologies. pp. 1–6. ICEET (2023). https://doi.org/10.1109/ICEET60227.2023.10525933
54. Yan, L., Greiff, S., Teuber, Z., Gašević, D.: Promises and Challenges of Generative Artificial Intelligence for Human Learning. Nature Human Behaviour **8**(10), 1839–1850 (Oct 2024). https://doi.org/10.1038/s41562-024-02004-5
55. Zeller, A.: Why Programs Fail: A Guide to Systematic Debugging. Elsevier Science (2009). https://doi.org/10.1016/B978-1-55860-866-5.X5000-0